CREARE UN ADD-IN
ALL'INTERNO DI VISUAL STUDIO .NET

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90
VERSIONE STANDARD RIVISTA+CD €6,90

Periodicità mensile • FEBBRAIO 2003 • ANNO VIII, N.2 (77)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioPROGRAMMO

Programmazione cellulari
Tim • Vodafone • Wind
TRE NUMERI IN UNA SIM
Passo passo come programmare una super scheda

- Leggi i tuoi dati
- Clona la tua carta sul computer
- Utilizza tre gestori diversi in un'unica SIM

IN ALLEGATO

SISTEMA
- WinZip in Java: l'interfaccia
- Le funzioni avanzate di JavaHelp

MULTIMEDIA
- Primi passi con DirectX e .NET

ELETTRONICA
- GPS: pilotarli con il protocollo NMEA

CORSI
- VB.NET: numeri, stringhe e date
- UML: i sequence diagram
- C++: gli algoritmi delle STL
- Delphi: tipi di dato
- Java base: i metodi

ADVANCED
- Alla scoperta della *Service Oriented Architecture*
- Il PC impara a giocare con il Pathfinding
- Strumenti e tecniche per il calcolo combinatorio

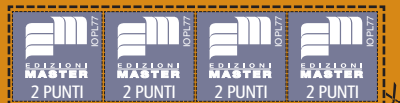
ISSN 1128-594X
ioProgrammo (Plus) Anno VIII - N° 2 (77) • € 9,90
EDIZIONI MASTER www.edmaster.it

XML IN PRATICA
- Ora è facile utilizzarlo anche in Java! Grazie alle nuove librerie JAXB
- La gestione di magazzino in C#

GESTIRE USB IN VISUAL BASIC 6
Il modo più semplice per controllare le periferiche

UN ROBOT CHE SERVE IL CAFFÈ
Come pilotare una mano meccanica con precisione

IN DIRETTA DA SUN: TUTTI I SEGRETI DELLE ETICHETTE INTELLIGENTI CHE STANNO CAMBIANDO IL MONDO

CONTENTS ▼ Questo mese su ioProgrammo

Anno VIII - n. 2 (77) Febbraio 2004

▼ Il futuro ha già l'etichetta

Questo mese mi sono occupato personalmente della traduzione di un interessante articolo pubblicato sul sito ufficiale di Sun Microsystem. Di taglio tecnico/divulgativo, l'articolo ha il pregio di fornire una rapida panoramica su una tecnologia che sarebbe riduttivo definire promettente: i tag RFID. Questi piccoli chip dotati di antenna andranno ad affiancarsi (e forse sostituiranno) i codici a barre che ormai contraddistinguono tutti i prodotti, ivi compresa la rivista che avete fra le mani. Simili concettualmente, i codici a barre ed i tag RFID differiscono sostanzialmente per tre aspetti: con i tag RFID non è necessario un contatto visivo per il riconoscimento; con l'RFID è possibile identificare il singolo oggetto, non solo la tipologia; le dimensioni ridottissime delle nuove etichette. Le sfide lanciate alla privacy da questi oggettini sono inquietanti. Tempo fa si era diffusa la voce che Benetton voleva includere, in ogni capo di abbigliamento, un tag non asportabile al fine di migliorare lo stoccaggio delle merci e di ridurre i rischi di taccheggio. Non asportabile. Vi rendete conto? Ogni volta che indosseremo una maglietta o un pantalone, saremo marchiati come deportati, come animali. Le nostre azioni e noi stessi potrebbero essere controllati come mai prima d'ora... questo per dire dei problemi etici, ma a noi è toccato il ruolo lieve di rivista tecnica. Dunque: state allerta, perché le possibilità che si aprono per noi sviluppatori con questo nuovo modo di marchiare la merce sono innumerevoli e difficilmente immaginabili. Che lavoriate in proprio o in azienda, molto probabilmente dovrete presto fare i conti con questa nuova e stimolante realtà. Sono certo che faremo grandi cose e, magari, invece di contribuire a costruire uno stato di polizia, potremo dare una mano a rendere la vita di tutti più semplice.

raffaele@edmaster.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **kane** *Password:* **rosebud**

ioProgrammo

Anno VIII - N.ro 2 (77) - Febbraio 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori S. Ascheri, M. Autiero, L. Buono, M. Camangi, M. Canducci, M. Del Gobbo, P. De Nictolis, G. Dodaro, F. Grimaldi, A. Marroccelli, M. Messina, F. Mestrone, G. Naccarato, C. Pelliccia, P. Perrotta, M. Popone, M. Scala, L. Salerno, D. Senatore, L. Spuntoni, D. Visicchio, F. Vaccaro; C. F. Zoffoli
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona

Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Rete Vendita: Serenella Scarpa, Cornelio Morari, Roberto Piano, John F. Alborante
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 37,90 sconto 50% sul prezzo di copertina € 75,90.
ioProgrammo Plus (11 numeri + 6 libri): € 61,90 sconto 50% sul prezzo di copertina € 123,90.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Gennaio 2004

Nessuna parte della rivista può essere in alcun modo riprodotta senza autorizzazione scritta della Edizioni Master. Manoscritti e foto originali, anche se non pubblicati, non si restituiscono. Edizioni Master non sarà in alcun caso responsabile per i danni diretti e/o indiretti derivanti dall'utilizzo dei programmi contenuti nel supporto multimediale allegato alla rivista e/o per eventuali anomalie degli stessi. Nessuna responsabilità è, inoltre, assunta dalla Edizioni Master per danni o altro derivanti da virus informatici non riconosciuti dagli antivirus ufficiali all'atto della masterizzazione del supporto. Nomi e marchi protetti sono citati senza indicare i relativi brevetti.

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, <tag/>, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection.

PARAFRASANDO

Al Massachusetts Institute of Technology di Boston, stanno sperimentando un software che riesce a parafrasare intere proposizioni scritte in inglese. Attraverso complessi algoritmi e mutuando tecniche statistiche dall'analisi genetica, alcuni ricercatori del MIT hanno messo a punto un programma che, data una frase, ricostruisce una sorta di pattern sulla cui base crea delle nuove frasi che, utilizzando parole diverse, restituisce la stessa informazione della frase di partenza. Le applicazioni ipotizzabili sono molteplici: dall'ausilio a chi si occupa di scrittura alla "riduzione" di libri ed enciclopedie per ottenere delle versioni "per bambini", una volta fissato un vocabolario adatto.

www.csail.mit.edu

IL GUFO APRE GLI OCCHI

L'acronimo del Web Ontology Language è "OWL" che, in inglese vuol dire appunto "gufo". La notizia è che, con largo anticipo sulle previsioni, OWL ha raggiunto lo status di Proposed Reccomendation. OWL è lo standard proposto dal World Wide Web Consortium per la definizione di una semantica del Web. Attraverso XML, è possibile descrivere la natura delle informazioni che si offrono, rendendo possibile delle modalità di ricerche finora impensabili. Sintetizzando, tra i risultati di questa tecnologia dovremmo avere dei motori di ricerca che rispondano a domande del tipo: "Mostrami i dipinti del periodo blu di Picasso in cui non siano presenti soggetti femminili".

www.w3.org/TR/owl-features/

News

INTEL, VODAFONE E ACER INSIEME PER IL WIRELESS

Un iniziativa congiunta dei tre colossi per proporre in Italia una soluzione di connettività "always on" alla clientela professionale. La partnership consentirà agli utenti in possesso di una Mobile Connect Card di Vodafone e di notebook o Tablet PC Acer basati sulla tecnologia mobile Intel Centrino di essere sempre connessi a Internet e alla rete aziendale o via GPRS oppure, attraverso un collegamento Wi-Fi, in presenza di un hot spot.

www.vodafone.it

DOCOMO SCOMMETTE SU LINUX

Forte del grande seguito che la tecnologia 3G sta avendo in Giappone, DoCoMo ha deciso di investire massicciamente nel settore e, assieme a Nec e ad altri grandi aziende, sta approntando i primi telefonini basati su tecnologia Linux. L'abbattimento dei costi derivanti dall'adozione di standard aperti consentirà all'azienda un interessante dirottamento degli investimenti previsti che, in buona parte, si concretizzerà nel supporto di telefonini WiFi che, in presenza di hot spot, commuteranno automaticamente la comunicazione su VoIP, a tariffe fortemente ridotte.

www.nttdocomo.com

MONO 1.0 DISPONIBILE ENTRO LA METÀ DEL 2004

Novell ha recentemente presentato la roadmap del progetto Mono, una iniziativa per lo sviluppo della versione open source della piattaforma Microsoft .NET che permetterà agli sviluppatori Linux e UNIX di creare e distribuire applicazioni .NET cross-platform.
La roadmap prevede il rilascio della versione 1.0 di Mono entro la prima metà del 2004, ed il rilascio successivo di ulteriori versioni che permetteranno agli sviluppatori di applicazioni aziendali e ai produttori software di creare applicazioni basate sulla piattaforma .NET su un'ampia varietà di piattaforme e sistemi operativi.
"Linux quale sistema operativo per il desktop sta diventando un'opzione sempre più appetibile per un

DIVORZIO FRA SUN ED ECLIPSE.ORG

Un'eclissi senza sole… il gioco di parole è banale ma la notizia è di quelle gravi: dopo mesi di incomprensioni fra i vertici IBM e quelli Sun, la decisione è stata in fine presa e non ci sarà più alcuna partecipazione da parte di Sun Microsystems nel progetto Eclipse. Tramontano per sempre i sogni che avevano previsto un'unica piattaforma di sviluppo open source per Java, la cui comunità sarebbe stata di gran lungo la più estesa del mondo della programmazione. NetBeans, il progetto omologo della Sun che avrebbe dovuto confluire in Eclipse, continuerà a svilupparsi sulle proprie gambe ed entro la fine del 2004 potremo vedere il concretizzarsi della prossima major release, la 4.0.
Anche in questo caso, le ragioni delle aziende dominanti hanno prevalso su quelle degli utenti/sviluppatori che avrebbero tratto grande beneficio dall'unione degli sforzi dei due colossi. Questa frammentazione non potrà che riflettersi negativamente sul processo di crescita delle piattaforme, a grave danno dei tantissimi entusiasti utilizzatori di Eclipse.

www.eclipse.org

numero crescente di acquirenti IT", ha commentato Chris Stone, Vice Chairman, Office of the CEO di Novell. "Per avere successo, gli sviluppatori hanno bisogno di un ambiente di sviluppo produttivo, di API stabili e di una roadmap tecnologica ben definita. Mono fornisce una risposta a tutte queste esigenze, oltre a portare i benefici di .NET agli ambienti Linux e UNIX".
Mono, al quale sono stati dedicati oltre 2 anni di attività di progettazione e testing, semplifica la creazione e la distribuzione di applicazioni su Linux e UNIX. Il progetto avanza costantemente, per supportare l'evoluzione delle tecnologie desktop Microsoft .NET, Linux e GNOME e la roadmap assicura a sviluppatori e produttori software di effettuare efficaci pianificazioni e di essere parte attiva del processo.

www.go-mono.com/

UNA PATCH OPEN SOURCE PER EXPLORER

Bruciando sul tempo gli sviluppatori di casa Microsoft, il sito Web Openwares.org ha pubblicato una patch che corregge una grave vulnerabilità di Internet Explorer che consentiva di visualizzare nel browser una pagina diversa rispetto a quanto riportato nella barra degli indirizzi. La patch si è dimostrata efficace, ma molti analisti sconsigliano la sua installazione in quanto potrebbe portare problemi di incompatibilità con le future patch distribuite da Microsoft.

www.openwares.org

NUOVO TOOL IBM PER AMMINISTRATORI DI DB

Sul sito della alphaWorks è apparsa la versione beta di un nuovo tool di IBM indirizzato ad amministratori e manager coinvolti nella gestione di database, ma che non abbiano skill specifici in materia. Policy Based Data Management tool, questo il nome del software che consentirà la definizione di complesse policy anche a chi non ha esperienza nel settore. Il tool disporrà di due interfacce: una per la definizione degli oggetti su cui le policy andranno ad operare, l'altra per la definizione delle policy vere e proprie. Il tool si farà carico di eseguire le varie policy e di segnalare eventuali conflitti fra policy diverse, già al momento della stesura.

www.alphaworks.ibm.com/

QUANDO IL GOVERNO CANCELLÒ LA PRIVACY

Con un decreto del 23 dicembre 2003 il governo ha deciso di cancellare di fatto la privacy dei cittadini che fanno uso di Internet. Il decreto legge obbliga gli operatori telefonici e i provider a conservare tutto il traffico generato dagli utenti per un periodo di cinque anni. Sms, mail e siti visitati entreranno a far parte di una enorme banca dati che creerà di fatto un società di schedati. Stefano Rodotà, garante della privacy, si è affrettato a denunciare la semplice evidenza che, una volta conosciuti i siti frequentati ed i contatti mail di una persona, è possibile risalire facilmente a dati sensibili quali il credo politico o lo stato di salute. Da molte forze politiche, e dai verdi in primo luogo, c'è stata una forte sollevazione anche alla luce dell'articolo 15 della costituzione che afferma l'inviolabilità della segretezza della corrispondenza e di ogni altra forma di comunicazione. Nell'occidente si assiste ormai ad una strage delle libertà personali in nome della guerra al terrorismo.

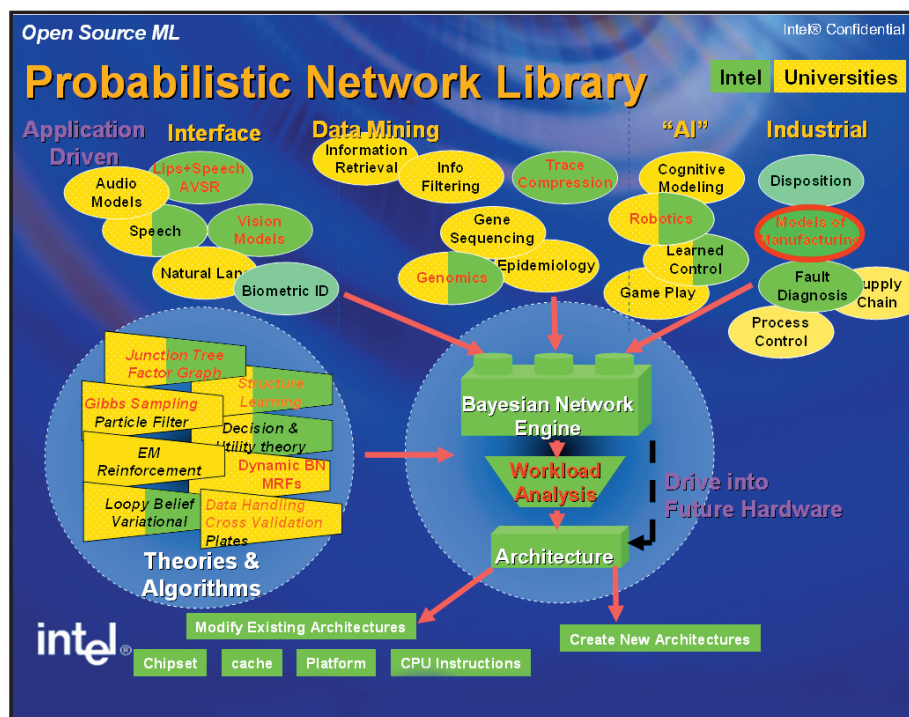
IL COMPUTER IMPARA

Intel appronta una nuova tecnologia che consente ai PC di imparare dalle proprie esperienze.

Con l'intento di coinvolgere la più ampia comunità possibile, la piattaforma è stata rilasciata come Open Source. La libreria alla base del progetto è la Probabilistic Network Library (PNL), basata sui concetti bayesiani del calcolo delle probabilità. Gli sviluppi futuri di un fenomeno vengono preconizzati sulla base di calcoli probabilistici, a partire dagli eventi già successi nel passato.

Il campo di utilizzo previsto da Intel è vastissimo e va dalla ricerca scientifica ai modelli comportamentali per i giochi dei bambini: grazie alle PNL, sarebbe possibile avere dei giochi che si adattino al comportamento dei bimbi. Le PNL vanno ad affiancarsi ad altre due librerie: Computer Vision Library e Audio-Visual Speech Recognition.

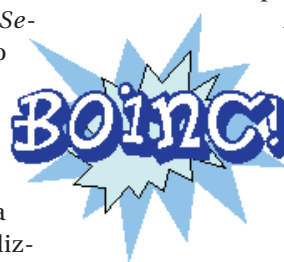
Queste tre librerie fanno parte di un unico progetto che mira a costruire una sorta di automa capace di interagire in modo più naturale con l'ambiente che lo circonda e di imparare dalle esperienze che si trova ad affrontare.

www.intel.com/research/mrl/pnl/

MICROSOFT SAREBBE PRONTA A PRESENTARE UN SO LINUX

Secondo eWeek.com Microsoft sarebbe pronta a relizzare un SO Linux nel caso in cui il mondo del pinguino diventasse un agguerrito concorente. Il sol fatto che Microsoft abbia pagato una licenza a SCO è un primo passo che lascia presagire, qualora il mercato lo richieda, di integrare alcuni servizi Unix nei suoi prodotti. I Services for Unix danno la possibilità, agli utenti Microsoft, di accedere ad un ambiente di shell Unix, che gira sul kernel di Windows. Sarebbe quindi possibile far "girare" un kernel Linux su di una macchina Windows 2003, non solo tecnicamente, ma anche legalmente, avendo Microsoft acquistato le licenze SCO.

SUN INVESTIRÀ IN SETI@HOME

Sun Microsystems ha siglato un accordo con la Università della California per collaborare alla prossima generazione del progetto *Seti@home*, noto fin dal 1999 per aver reso possibile su larga scala l'utilizzo della potenza inutilizzata dei PC. Grazie ad uno screensaver che chiunque può scaricare, milioni di PC hanno donato alla ricerca scientifica la loro potenza di calcolo al fine di scoprire l'esistenza di forme di intelligenza extraterrestri. Il progetto *Seti@home* è costato finora mezzo milione di dollari, fornendo una potenza di 15 TeraFLOPS a fronte dei 12 TeraFLOPS disponibili con un IBM *ASCI White system*, il cui costo è di 110 milioni di dollari. La seconda generazione di *Seti@home* rientrerà in un progetto più grande, il *Berkeley Open Infrastructure for Network Computing* (BOINC) che consentirà di sfruttare le risorse condivise per uno spettro di ricerche più ampio rispetto alla sola ricerca di intelligenze extraterrestri.

http://boinc.berkeley.edu/

SOFTWARE SUL CD

Una selezione dei migliori tool di sviluppo proposta dalla redazione di ioProgrammo

Eclipse 3.0 M5

Il più potente framework open source per la programmazione.

di Massimo Canducci

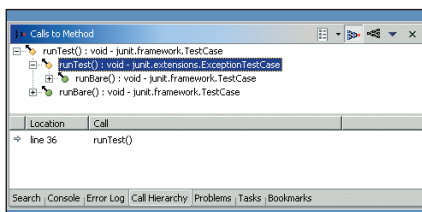
Da qualche mese il consorzio Eclipse (*www.eclipse.org*) sta lavorando alacremente nella produzione della release 3.0 di Eclipse, il potente ambiente di sviluppo Java oriented ed open source. Eclipse è in realtà una piattaforma integrata per lo sviluppo, il debug ed il test di applicazioni enterprise la cui architettura di base, dal valore iniziale di circa 40 milioni di dollari, è stata donata da IBM alla comunità Open Source. Il senso di Eclipse è quello di fornire agli sviluppatori degli strumenti di sviluppo sempre più sofisticati e potenti grazie ad un unico framework che integri potenzialmente tutti i linguaggi, gli Ide, i tool di modeling e gli strumenti di collaborazione esistenti. Si tratta di un framework basato su un sistema di API pubbliche e di plug-in le cui specifiche di sviluppo sono di pubblico dominio e questo permette a chiunque di sviluppare i propri plugin e renderli disponibili alla comunità Open Source oppure crearne versioni particolari a pagamento. L'ultima versione stabile di Eclipse, quella che viene utilizzata da moltissimi sviluppatori in tutto il mondo, è la 2.1.2 scaricabile gratuitamente dal sito del consorzio ed ha qualità e caratteristiche di produttività, per la realizzazione di applicazioni Java Enterprise, fino a poco tempo fa inimmaginabili, anche su prodotti di mercato dal prezzo elevato. Trattandosi di un progetto curato da una comunità Open Source, Eclipse continua inarrestabile la sua crescita, da qualche tempo ormai la comunità sta lavorando alla versione 3.0 ed il calendario prevede 9 diverse milestone con le quali le funzionalità più importanti vengono congelate e rilasciate come versioni beta, l'ultima milestone, la M9, dovrebbe essere rilasciata a Maggio 2004, quindi nella seconda metà del prossimo anno dovrebbe essere rilasciata definitivamente la versione 3.0 stabile (Fig. 1).

Le milestone intermedie sono scaricabili liberamente ed il 21 Novembre è stata rilasciata la M5 che permette già di apprezzare un notevole numero di nuove funzionalità e caratteristiche, quello che segue è un elenco, purtroppo non esaustivo, delle funzioni più importanti presenti nella M5.

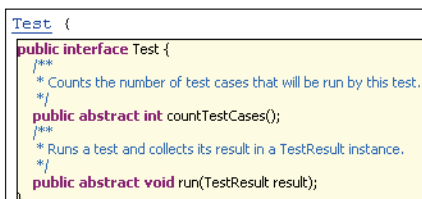
***Fig. 1:** Una vista complessiva della nuova interfaccia.*

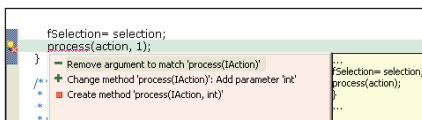
LE NUOVE FUNZIONALITÀ DELLA M5

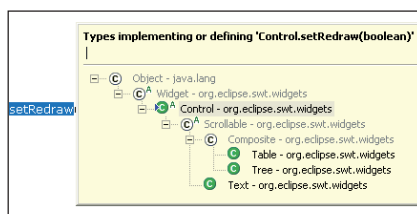
Vista gerarchica sulle chiamate ad un metodo

nel menu *Navigate* esiste ora la voce *Open Call Hierarchy* che consente di visualizzare una finestra con tutte le

informazioni gerarchiche riguardo tutte le chiamate ad un metodo.

Tooltips colorati

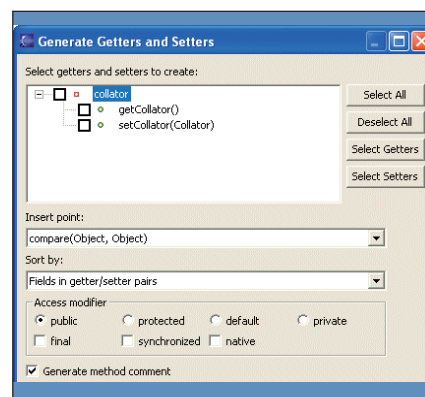
i tooltips contenenti codice sorgente hanno una colorazione del codice identica a quella del codice sorgente.

Nuovi Quick Fix

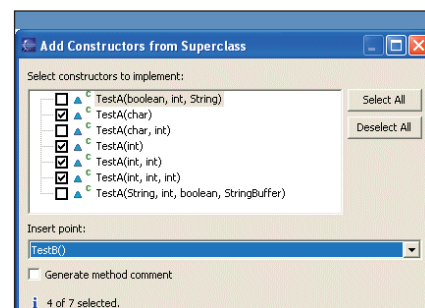
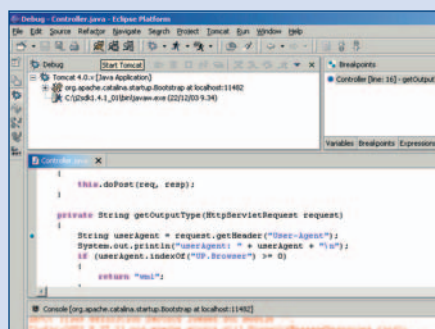
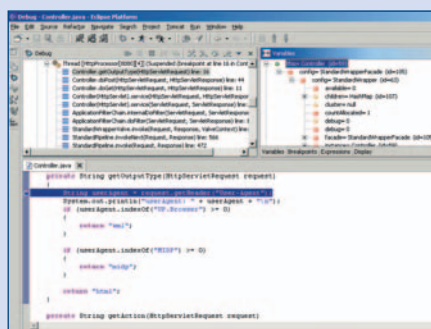
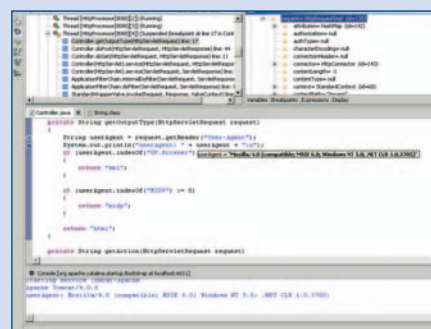
sono state aggiunte nuove funzionalità di quick-fixing oltre alle ottime già disponibili, in figura uno strumento per la correzione rapida sulla chiamata di un metodo con la possibilità di rimuovere al volo i parametri eccedenti di un metodo oppure creare al volo un nuovo metodo con la firma identica alla chiamata che si tenta di eseguire.

Quick Type Hierarchy View

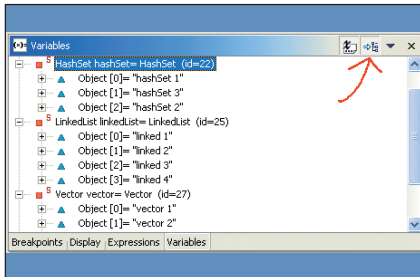
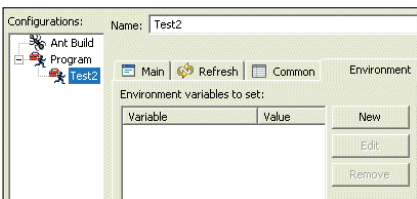
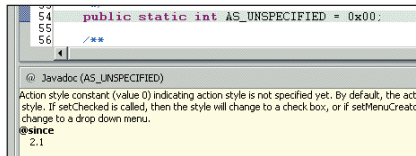
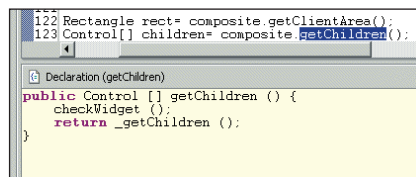
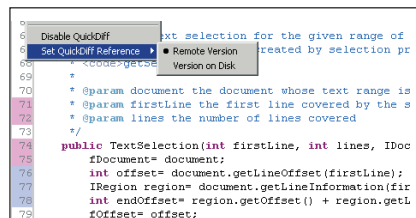
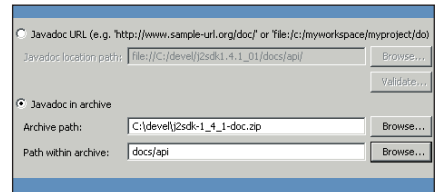
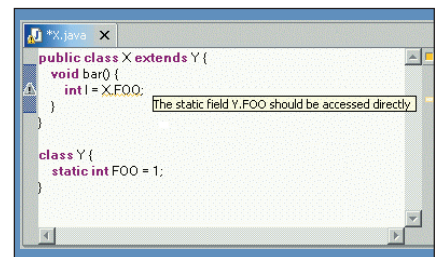
posizionandosi su di un oggetto e premendo la combinazione di tasti CTRL-T si ottiene l'apertura di un tool-tip contenente tutta la gerarchia completa che porta a quell'oggetto. Se ci si posiziona su un metodo vengono anche mostrati tutti i supertipi ed i sottotipi che lo forniscono.

Miglior integrazione con JUnit

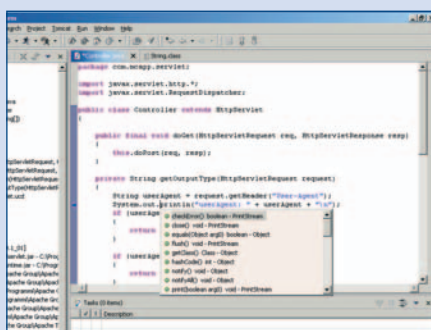
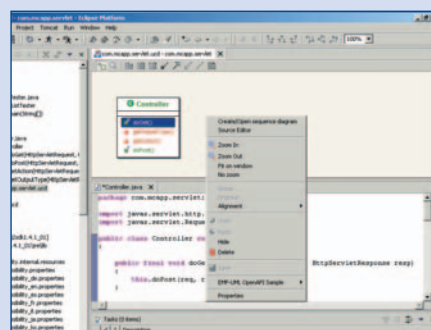
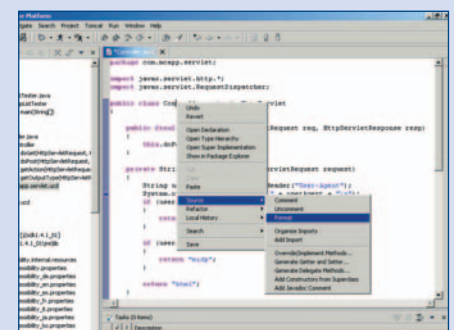
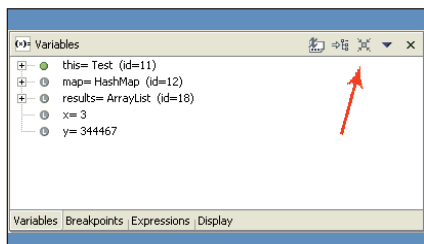
il framework per la gestione completa dei test unitari JUnit è sempre stato ben integrato con Eclipse, in questa release l'integrazione si fa più stretta e più usabile.

Generazione automatica del codice

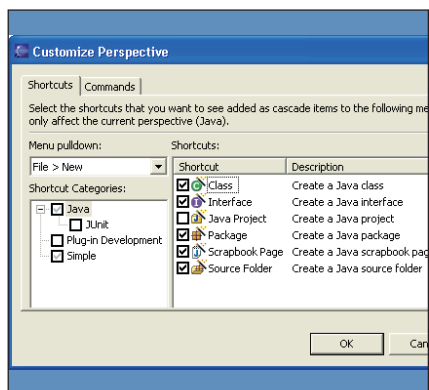
le funzionalità di generazione automatica del codice sono state migliorate, adesso viene fornito un controllo maggiore su cosa viene generato e come.

Generazione automatica dei costruttori

si tratta di un'ottima funzionalità che consente, all'interno di una sottoclasse, di generare automaticamente il codice di override dei costruttori, in questa versione viene mostrato un popup che consente di selezionare quali costruttore riscrivere nella classe attuale.

Il debugging

1 **Durante lo sviluppo di un'applicazione web J2EE compliant è possibile eseguirla in modalità debug utilizzando il servlet engine di Tomcat. È sufficiente far partire Tomcat dagli strumenti dell'apposito plugin e l'ambiente passerà immediatamente alla modalità debug.**

2 **Durante l'esecuzione del codice il puntatore si ferma sul breakpoint che è stato attivato e si hanno tutte le funzionalità tipiche di un ambiente di debug. In alto a destra tutte le informazioni riguardo i breakpoint attivi, le variabili nel contesto, eventuali espressioni che possono essere valutato online.**

3 **Durante le operazioni di debug è possibile avere informazioni immediate sul valore delle variabili grazie a comodi tooltips. Nella vista in alto a sinistra compare la lista di tutti i thread dell'applicazione che sono in esecuzione nell'ambiente dell'application server.**

Filtri sulle variabili
all'interno del debugger è possibile gestire gerarchicamente le variabili, in particolare se si utilizzano collezioni e maps queste possono essere visualizzate in modalità gerarchica.

Variabili d'ambiente per software esterni
la gestione dei software esterni diventa più flessibile grazie alla possibilità di definire variabili d'ambiente da inizializzare nella shell che li esegue.

Javadoc View
all'editor è stata aggiunta una nuova vista che consente di avere sempre a disposizione le informazioni presenti nei commenti Javadoc degli oggetti e dei metodi che si stanno utilizzando.

Declaration View
all'editor è stata anche aggiunta una vista che consente di avere a disposizione il codice sorgente del metodo che si sta utilizzando.

Quick Diff
nell'editor è stato inserito uno strumento che segnala quali sono le modifiche che sono state effettuate sul codice sorgente in funzione di quanto presente sul file salvato in precedenza, queste modifiche possono poi essere selettivamente annullate o modificate.

Gestione dei Javadoc compressi
se si ha a disposizione un javadoc compresso non è più necessario compattarlo ma è sufficiente utilizzare l'opzione in figura per avere a disposizione la documentazione.

Accesso indiretto a membri statici
il compilatore è ora in grado di segnalare il verificarsi di un accesso indiretto ad un membro statico di una classe.

Visualizzazione delle variabili
durante la fase di debug le variabili

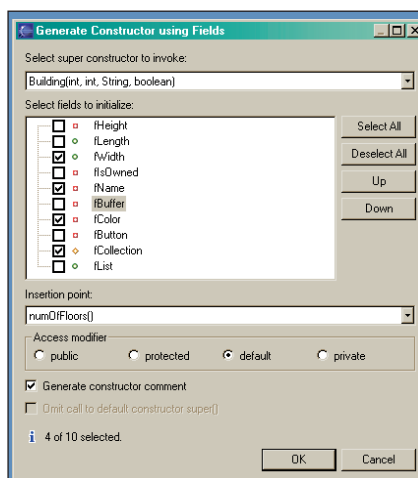
La stesura del codice

1 Durante la stesura del codice Eclipse presenta i classici strumenti di produttività, a cominciare dalle comode tendine che presentano la lista di tutti i metodi e gli attributi di un oggetto che si sta utilizzando.

2 Lo sviluppo può essere fatto, grazie ad appositi plugin, affiancato alla progettazione UML. In figura un class-diagram realizzato automaticamente dal plugin della Omondo, a partire da un codice sorgente già disponibile.

3 Le operazioni di editing sono facilitate anche da un insieme molto completo di strumenti, tra questi un comodo *formatter* che può essere configurato in modo da disaccoppiare la semantica dalla sua rappresentazione sintattica.

possono essere collassate ed espanse grazie ad una serie di nuovi pulsanti della vista.

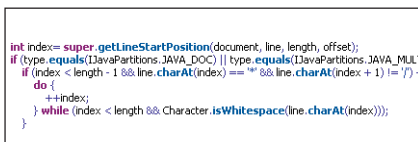
Personalizzazione delle prospettive

il nuovo ambiente permette una personalizzazione più spinta delle varie prospettive a disposizione, è possibile personalizzare, oltre alle viste, anche le voci di menù ed i comandi eseguibili in una singola prospettiva.

Nuovi wizard per creare i costruttori

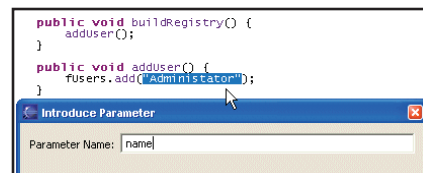
utilizzando il wizard in figura è possibile creare i costruttori delle classi inizializzando automaticamente i campi selezionati.

Nuova syntax highlighting

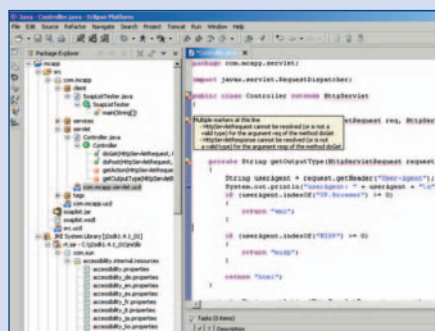
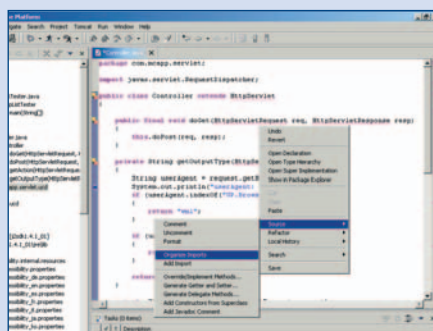
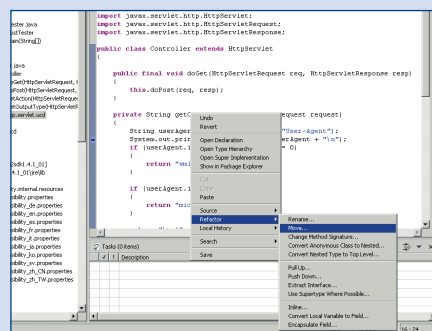
il nuovo Java Editor è in grado di colorare i nomi dei metodi.

JDK 1.4

il compilatore Java funziona in due modalità, una compatibile con il JDK 1.4a e l'altra compatibile con il JDK 1.3.

```
int index= super.getLineStartPosition(document, line, length, offset);
if (type.equals(IJavaPartitions.JAVA_DOC) || type.equals(IJavaPartitions.JAVA_MULT
   if (index < length - 1 && line.charAt(index) == '*' && line.charAt(index + 1) != '/') {
      do {
         ++index;
      } while (index < length && Character.isWhitespace(line.charAt(index)));
   }
```

Refactoring

selezionando un'espressione ed utilizzando la funzione *"Introduce Parameter"* dal menù *Refactoring*, si ottiene l'inserimento di una nuova variabile che viene aggiunta ai parametri in ingresso del metodo corrente.

Queste sono solo alcune delle nuove potenti funzioni della versione 3.0 di Ecplise e sono già disponibile con la M5. Le varie versioni di Eclipse si possono trovare a partire dal link *http:// www.eclipse.org*. Un vasto numero di plugin open source possono essere scaricati a partire da: *http://www.eclipse-plugins.info*.

> **☑ Eclipse 3.0 M5**
> **Produttore: consorzio eclipse.org**
> **Sul web: www.eclipse.org**
> **Prezzo: gratuito**
> **Nel CD: \Open\Eclipse**

Refactoring

1 Durante la stesura del codice possiamo utilizzare oggetti Java anche senza averli correttamente importati. Nel nostro caso, ci sono una serie di errori di compilazione, che avviene contemporaneamente alla stesura del codice.

2 La funzione *Organze Imports* consente di inserire automaticamente tutti gli import che ci siamo dimenticati e di eliminare quelli eventualmente presenti ed inutili, liberando lo sviluppatore dall'onere di apportare le modifiche manualmente.

3 Eclipse contiene un numero impressionante di strumenti di refactoring. In questo esempio possiamo spostare o rinominare una classe o un metodo, tutti i riferimenti all'oggetto risulteranno aggiornati.

3D Developer Studio Pro 6.02

Creare videogiochi al top

La terza dimensione entra nelle nostre applicazioni e lo fa attraverso uno strumento di livello professionale che consente a chiunque di avvantaggiarsi di tutte le più avanzate tecniche di 3D. 3D Developer Studio Pro supporta direttamente alcuni fra i più diffusi linguaggi di programmazione: Microsoft Visual C++, Microsoft Visual C# .NET, Borland C++ Builder, Borland Delphi, Microsoft Visual Basic 6, Microsoft Visual Basic .NET, praticamente tutti, eccetto Java. Utilizzabile sia per lo sviluppo di videogiochi che per l'integrazione di oggetti tridimensionali nella GUI delle nostre applicazioni, la qualità del tool è testimoniata dai nomi che hanno scelto di adottarla: Borland, Motorola, la BBC, Computer Associates, IBM. Nel pacchetto trovate un cad per la costruzione di ambienti tridimensionali (*3D_WorldBuilder60.exe*) ed una serie di pacchetti di installazione, uno per ogni linguaggio, basterà scegliere quello adatto al compilatore che abitualmente usiamo.

Fig. 1: L'ambiente di sviluppo è ricco di pulsanti ma, già dopo I primi istanti, si riesce a lavorare con una buona agilità.

Fig. 2: Per ogni oggetto è possibile tenere sotto controllo tutti gli elementi che lo compongono.

SEMPLICE

Ineguagliabile la semplicità con cui è possibile trascinare negli ambienti che creiamo uno dei tantissimi oggetti tridimensionali già presenti in archivio.
Anche la scalatura e l'orientamento degli oggetti avviene sempre con la semplicità del drag&drop.
Il posizionamento delle telecamera ed il suo orientamento sono operazioni semplificate dalla simultanea presenta di tre punti di vista, due ortogonali e l'altro in prospettiva: in qualsiasi momento si ha sempre chiara la percezione di quale sia la nostra posizione all'interno dell'ambiente.

QUALITÀ

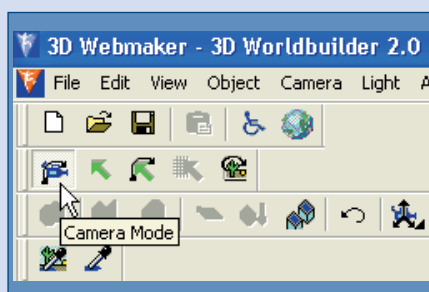
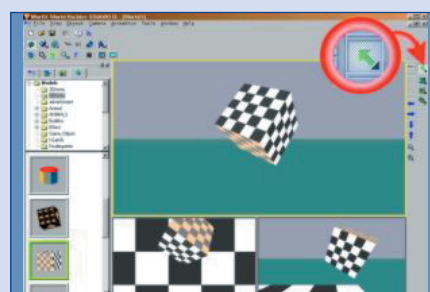
Pur non raggiungendo la qualità tipica delle soluzioni più avanzate, il motore di rendering di 3D Developer Studio offre comunque un livello più che sufficiente nella maggior parte degli ambiti applicativi. Livello che sale decisamente se il PC è dotato di una scheda 3D.

Fig. 2: Un museo virtuale online: notate la differenza qualitativa a seconda che sia attivato o meno l'accesso all'accelerazione 3D della scheda video.

PER TUTTI

Ciò che differenzia questo prodotto dalla maggior parte dei cad 3D esistenti è il fatto

Per cominciare

1 Per creare un nuovo mondo, cliccare sul pulsante "New" e scegliere uno dei template che vengono presentati: saranno la base su cui costruire i nostri ambienti.

2 Cliccando sul pulsante "Camera mode" è possibile spostare il punto di vista sia con i tasti di direzione che con il mouse. Tenendo premuto il tasto ctrl, la telecamera ruota.

3 Per selezionare un oggetto, è necessario cliccare sul pulsante "Object Movement Tool", e quindi cliccare sull'oggetto. Tenendo premuto il tasto alt, si può spostare l'oggetto stesso

di consentire la produzione di ambienti tridimensionali ad un vasto numero di utenti. Nessun prerequisito è richiesto e, la facile integrazione in siti Web e progetti software già esistenti, ne fa il software adatto ad aggiungere un tocco tridimensionale, con poco sforzo, in ambiti non pensati per la terza dimensione.

IL GIUDIZIO

Se volete farvi un'idea delle qualità di 3D Developer Studio, prima ancora di installarlo, potete dare un'occhiata ad uno dei numerosi esempi online che trovate elencati nel sito ufficiale www.3dstate.com. Dai giochi online ai negozi virtuali, fino ad arrivare ad un interessante esperimento di museo tridimensionale online, applicato addirittura al Louvre.

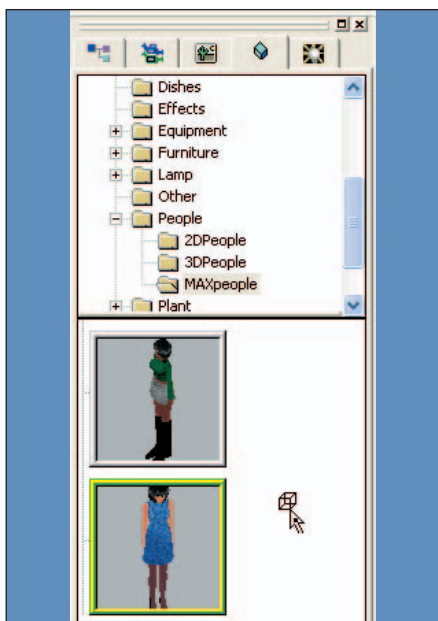



Fig. 3: *Davvero molto ricco l'archivio di oggetti 3D subito disponibili per l'inserimento in ambiente.*

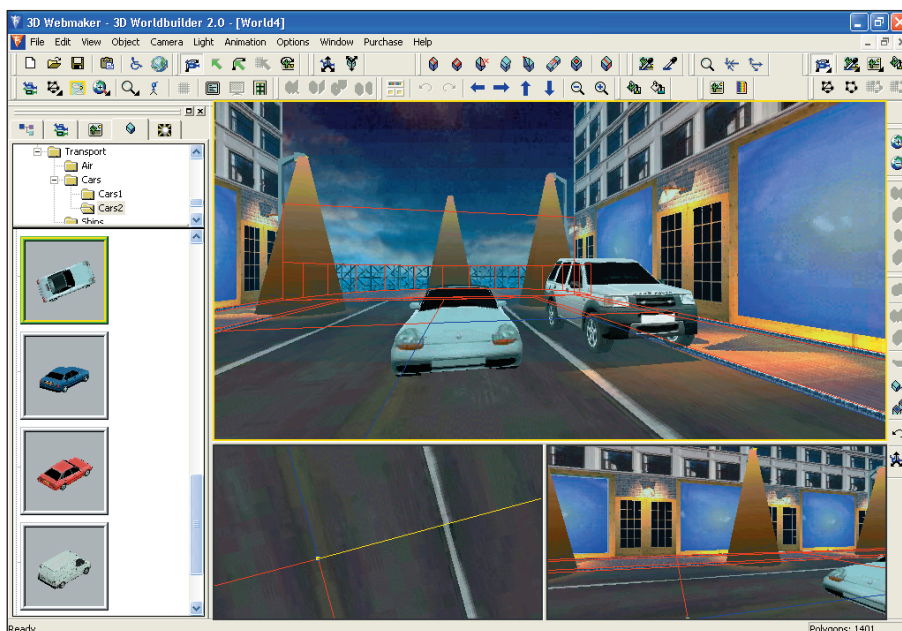



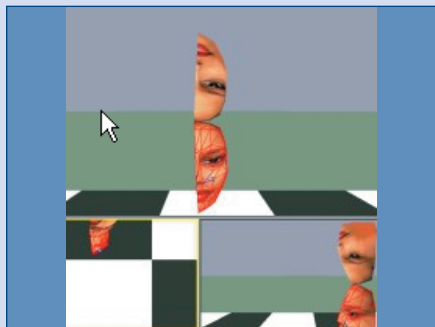
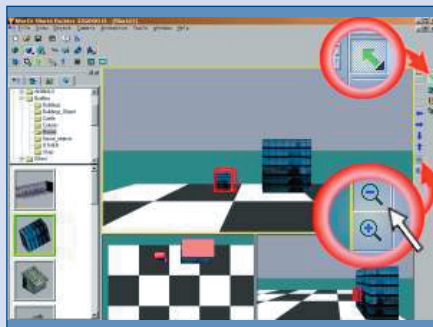
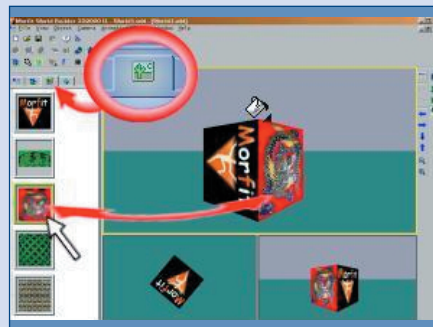
Fig. 4: *Anche per gli esterni ci sono molte possibilità di "arredamento"*

INSTALLAZIONE

Questa versione ha licenza gratuita, se utilizzata in progetti non commerciali. Per l'installazione è necessaria una chiave di attivazione che si può richiedere collegandosi a www.3dstate.com e cliccando sul link "Get A Free Serial number", presente nella parte superiore della pagina.
^3D_Developer_Studio_Pro.zip.

☑ 3D Developer Studio Pro 6.02

Produttore: 3D STATE Corporation
Sul web: www.3dstate.com
Prezzo: Gratuito
Nel CD: 3D_Developer_Studio_Pro.zip

Manipolare gli oggetti

1 Per effettuare il mirror di un oggetto è necessario che sia selezionato l'Object Movement Tool. Un clic con il tasto destro sull'oggetto e (premendo contemporaneamente X, Y o Z, per indicare l'asse) scegliere la voce "Create mirror".

2 Se si vuole cambiare la dimensione di un oggetto, è sufficiente selezionarlo (avendo preventivamente attivato l'Object Movement Tool) e cliccare sulla lente con "+" per ingrandirlo e su quella con "-" per ridurne la dimensione.

3 Al fine di aggiungere una texture ad un oggetto, selezionare "*Bitmap/Animation View*" nel pannello di sinistra, scegliere la figura da utlizzare e indicare il poligono o la superficie da riempire, così come indicato in figura.

Lutz Roeder's Reflector

Gli assembly .NET non avranno più segreti!

di Lorenzo Barbieri

Reflector è un tool sviluppato da Lutz Roeder (*www.aisto.com/roeder/dotnet*) che svolge molteplici funzioni, tutte molto utili:

- class browser
- decompilazione di metodi
- visualizzazione del codice IL dei metodi (disassembler)
- outline di tutti i metodi, membri, proprietà, etc... dell'elemento selezionato
- visualizzazione dei commenti XML e della documentazione MSDN dei metodi (se disponibili)
- ricerca di metodi e tipi
- ricerca delle referenze a metodi e tipi

Scaricando il programma (in formato .zip) si hanno a disposizione due versioni, la versione compilata con il .NET Framework 1.0 e la versione compilata con la versione 1.1
Eseguendo il programma ci si trova davanti alla schermata principale, dove è possibile vedere gli assembly caricati di default (quelli di sistema) e caricare gli assembly che ci interessa analizzare.
Tutti gli assembly caricati vengono visualizzati in un tree view, in cui è anche possibile navigare la gerarchia delle classi, sia verso i tipi base, sia verso i sottotipi (Fig.1).

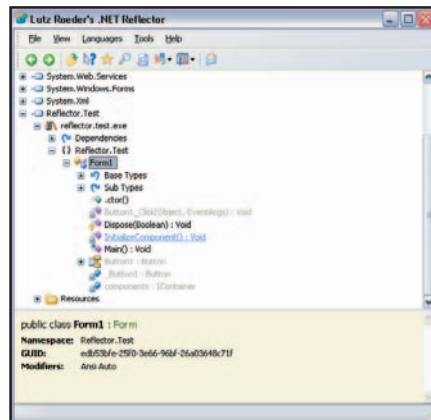



Fig. 1: *La geranrchia delle classi.*

DECOMPILAZIONE

La decompilazione permette di vedere il sorgente dei metodi, utilizzando i linguaggi C#, VB.NET e Delphi.
Naturalmente non è possibile vedere il sorgente dei metodi offuscati o contenenti codice nativo, ma la funzione risulta utilissima per capire il comportamento di metodi e classi di cui non si dispone il sorgente.
Attraverso questo tool è possibile decompilare anche moltissime classi della Base Class Library di .NET, per capirne il comportamento interno (Fig. 2). Questa funzione è molto comoda anche in quelle situazioni in cui non si ha più accesso ai sorgenti delle proprie applicazioni (come i code-behind delle applicazioni web di cui si sono persi i sorgenti) e si vuole ricostruirli.

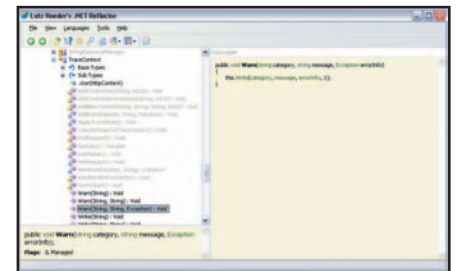
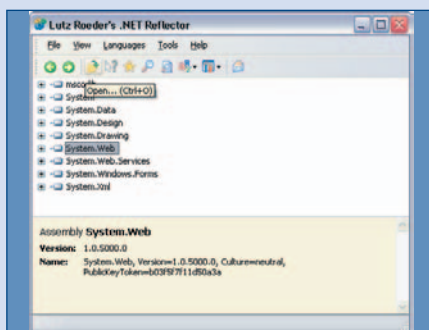
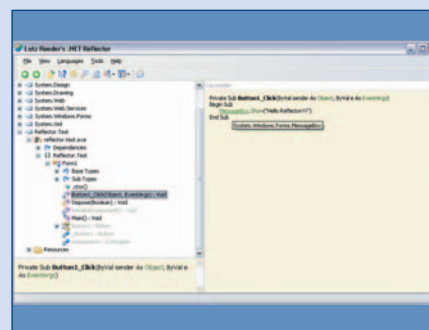
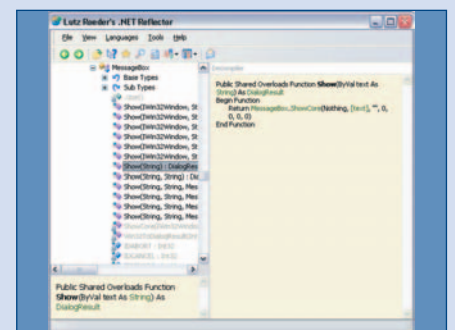



Fig. 2: *Decompilazione di classi.*

DOCUMENTAZIONE

La documentazione delle altre funzioni è molto scarsa (una pagina HTML), ma il tool è molto semplice da usare. In tutte le finestre che mostrano i dettagli sul codice (outline, decompilazione, disassemblati, etc...) tutti i metodi, le classi, etc... sono linkati e navigabili, facilitando molto la comprensione della struttura e delle relazioni tra di loro.

Visualizzare il codice di un metodo

1 **Aprire la dll o l'exe contenente il metodo (o i metodi) che ci interessa.**

2 **Navigare la gerarchia di classi fino al metodo che ci interessa e premere "SPAZIO" o tasto *dx -> Decompiler.***

3 **Ispezionare il codice del metodo, navigando eventualmente nelle parti attive (in verde).**

UTILIZZO DI REFLECTOR COME ADD-IN DEL VISUAL STUDIO .NET 2003

È possibile utilizzare Reflector come Add-In per Visual Studio .NET 2003 attraverso uno strumento di gestione degli Add-In chiamato *Managed Add-Ins* (*www.managedaddins.net*). Questo strumento permette di trasformare qualsiasi programma .NET in un add-in per il Visual Studio, trasformando le normali finestre Windows in finestre dockable integrabili nell'IDE (Fig. 3). Questo tool è molto configurabile e permette di richiamare creare nuovi menù contestuali e di associarli a particolari funzionalità dei programmi che ospita, dando l'impressione di una più stretta integrazione tra questi e il Visual Studio .NET. Oltre al Reflector esistono molti altri tool utilizzabili come AddIn, per una lista parziale si può vedere *www.managedaddins.net/ gallery*. Oltre a poter usare Reflector come Add-In, bisogna dire che anche Reflector stesso permette l'utilizzo di Add-In, ed è pure disponibile un SDK per la loro realizzazione, scaricabile dal sito internet.

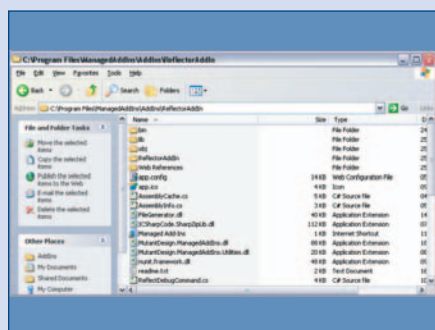
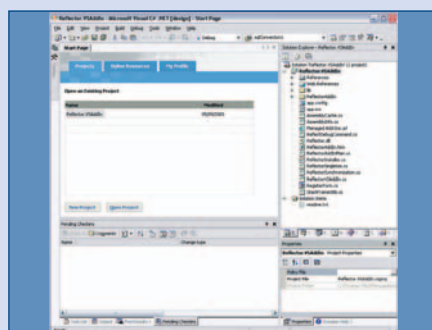
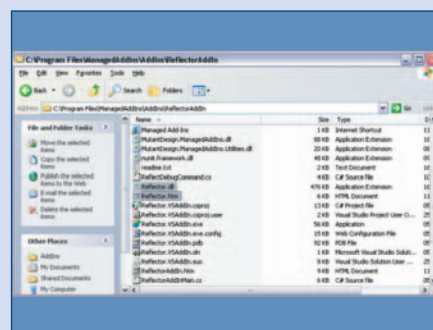
Fig. 3: Ecco come si presenta Reflector all'interno di Visual Studio .Net 2003.

CONCLUSIONI

Utilizzando Reflector ci si accorgerà ben presto di non poterne fare più a meno, soprattutto all'interno del Visual Studio.NET. Sul sito di Lutz Roeder sono disponibili anche molti altri tool, tutti molto interessanti, tra cui:

- **Resourcer.Net** – un editor di risorse binarie (.resources) e XML (.resx)
- **Documentor.Net** – un editor per i commenti XML inseribili nel codice C#, J# e VB.NET (quest'ultimo solo tramite appositi plug-in)
- **DatePicker.Net** – implementa un controllo calendario drop-down come quello di Outlook.
- **CommandBar.Net** – permette di implementare Command Bar, ReBar, e menu con BitMap (come quelli di Office).

☑ **Reflector**
Produttore: Lutz Roeder
Sul Web: www.aisto.com/roeder/dotnet/
Prezzo: Gratuito
Nel CD: \reflector

Utilizzare il tool Managed Add-Ins con il Reflector

1 Installare il tool Managed Add-Ins. Scompattare l'archivio *ReflectorAddIn-0.11.zip* nella cartella del *Reflector Add-In* creata dal setup precedente, sovrascrivendo i file.

2 Aprire con il VS.NET i sorgenti del Reflector AddIn ed eseguirlo. Il tool scaricherà il Reflector da Internet e lo installerà nella cartella consentendo di utilizzare l'add-in in Visual Studio.

3 In alternativa, scompattare la versione del Reflector per il proprio ambiente (1.0 o 1.1), rinominare *Reflector.exe* in *Reflector.dll* e copiarlo nella cartella dell'Add-In assieme al file *Reflector.htm*.

Installer2Go 3.2.1

Un sistema facile e gratuito per realizzare pacchetti di installazione

Un tool per la creazione di file autoinstallanti che non impegna lo sviluppatore nel dover imparare l'ennesimo linguaggio di scripting: con una interfaccia che punta tutto sul drag&drop, realizzare pacchetti di installazione diventa un vero gioco da ragazzi. Benché gratuito, Installer2Go va incontro a tutte linee guida per la certificazione WindowsXP. Installer2GO include una ottima utility, *Installer2GO.com:* una applicazione a riga di comando che può essere invocata da una finestra DOS o essere inserita in un file batch all'interno di un processo di installazione. La sintassi è:

```
[Installer2GO path]\Installer2GO.com"
    -build "[project file path]\My project.i2g.
```

È importante evitare di utilizzare *In-*

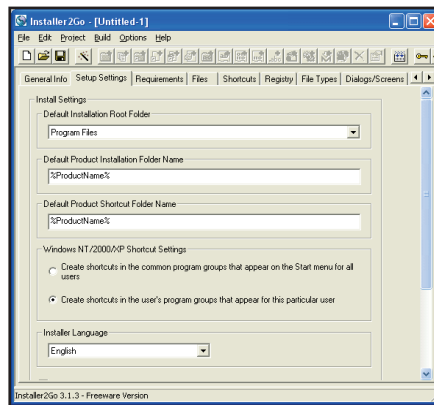



Fig. 1: *Pochi passaggi e l'installer è pronto.*

staller2GO.exe dalla linea di comando e all'interno di un file batch. È necessario separare i parametri con uno spazio e, nel caso in cui alcuni path contengano degli spazi, diventa obbligatorio racchiudere i path tra doppi apici. Alla fine di ogni istallazione si è rimandati ad una pagina pubblicitaria che fa riferimento al produttore di Installer2Go. Accettando questa limitazione, Installer2GO può essere utilizzato come Freeware.

Acquistando invece una licenza, è possibile rimuovere i messaggi promozionali che vengono visualizzati durante l'installazione dei pacchetti. Il costo di una chiave di attivazione è pari a $49.00 ma, acquistando più di venticinque licenze, è possibile ottenere degli sconti congrui.

☑ Installer2Go 3.2.1
Produttore: SDS Software
Sul Web: www.dev4pc.com
Prezzo: $49.00
Nel CD: i2g.exe

QuickXML 1.02

Tre differenti interfacce per creare e modificare file XML.

Innovativo e personalizzabile, QuickXML consente di gestire i file XML in modo ottimale grazie soprattutto alla disponibilità di tre differenti viste tra cui l'utente può scegliere: Tree View, Source View, e Browser View. La Tree View è quella che presenta maggiori doti di originalità: accanto alla

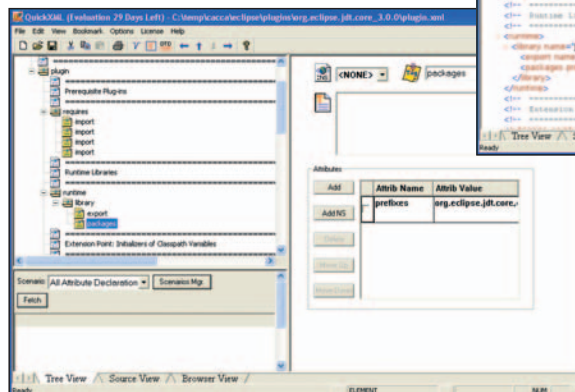
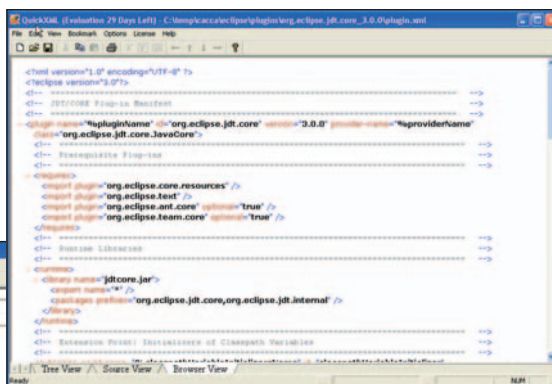



Fig. 1: *Elementi, testi e attributi in un'unica finestra di dialogo.*

vista ad albero, che sottolinea la struttura dei dati presenti nel documento XML, avremo a disposizione numerosi controlli per automatizzare l'inserimento e la modifica dei dati. È compreso il supporto per i namespace ed è molto utile la possibilità di inserire tutti gli elementi, il testo e gli attributi attraverso un'unica finestra di dialogo. È possibile editare file XML, XSL, XSD e DTD. Molto apprezzabile lo sforzo di originalità che permea l'interfaccia e che potrà incontrare i favori di chi si trovi a svolgere pesanti mansioni di data-entry. Più difficile immaginare QuickXML in un ambiente di sviluppo in cui, un framework più solido ed una interfaccia dall'utilizzo più immediato è probabilmente preferibile. Versione di prova valida trenta giorni.

☑ QuickXML 1.02
Produttore: SoftBeyond
Sul Web: www.softbeyond.com
Prezzo: $35.00
Nel CD: QX102Setup.exe

Realbasic 5.2.4

Crea e compila applicazioni per Windows e Mac.

Un ambiente di programmazione che rende disponibile, anche ai meno esperti, la possibilità di sviluppare applicazioni in pochissimo tempo, grazie anche alla ricca documentazione, ai numerosi tutorial e agli esempi inclusi. Oltre ad offrire la possibilità di importare codice e form da Visual Basic, Realbasic consente di compilare le applicazioni sviluppate, oltre che per Windows, anche per Mac OS 8, Mac OS 9 e Mac OS X. In questa minor release sono stati introdotti numerosi miglioramenti, il più importante dei quali riguarda l'estrema riduzione dei tempi di compilazione. Gli utenti Visual Basic di Microsoft non si troveranno disorientati, grazie ad una esplicita aderenza di Realbasic alle consuetudini della casa di Redmond nelle interfacce, senza contare che la migrazione per gli utenti Visual Basic è facilitata anche da un apposita utilità denominata *VB Project Converter*. Chi lavora in team potrà beneficiare del Project Manager che consente a più utenti di collaborare sullo stesso progetto. Versione dimostrativa valida dieci giorni. Al primo avvio è necessario cliccare su *"Get a demo Key"* per ottenere una chiave valida.

Fig. 1: Consente di importare codice e form da Visual Basic.

Realbasic 5.2.4
Produttore: REAL software
Sul Web: www.realsoftware.com
Prezzo: $159.95
Nel CD: REALbasicDemoSetup.exe

WinDriver 6.03

Genera il codice dei driver automaticamente.

Un tool che consente di accedere alle periferiche Hardware, senza la necessità di scrivere il codice dei driver. Tutto l'hardware presente sulla macchina viene identificato automaticamente attraverso una semplice interfaccia grafica, dopo di che WinDriver si occupa di generare lo "scheletro" del codice del driver, già ottimizzato per l'hardware individuato.

Particolarmente utile il debugger integrato che consente di monitorare l'attività del sistema in kernel mode.

Il DriveWizard ci guida in tutte le fasi della costruzione, consentendo anche ai meno esperti di prendere confidenza con l'ambiente e le problematica insite nella creazione di driver. Molto interessanti gli esempi presenti nel pacchetto di installazione che possono essere la base per cominciare nuovi progetti senza dovere partire da zero. Gli esempi riguardano molti importanti produttori di periferiche: PLX, Altera, Cypress, QuickLogic, National Semiconductor, STMicroelectronics, Texas Instruments, Xilinx, PLDA e AMCC. Le periferiche supportate sono moltissime, tra i produttori si annoverano: PLX, Altera, Cypress, QuickLogic, National Semiconductor, STMicroelectronics, Texas Instruments, Xilinx, PLDA e AMCC. Versione di valutazione valida trenta giorni.

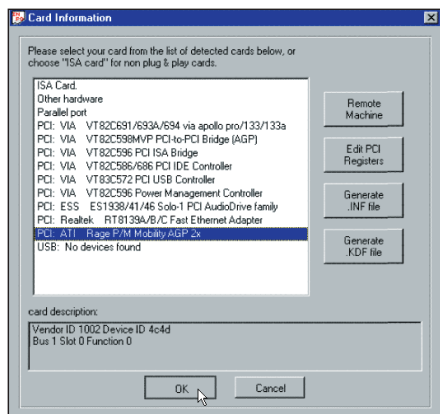
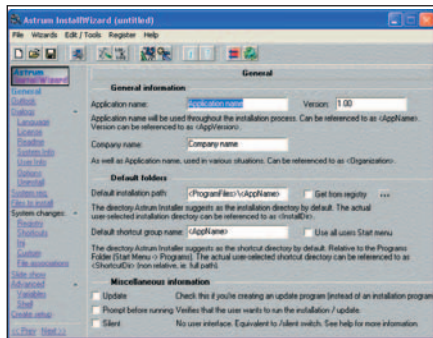



Fig. 1: Da notare la funzione di Debugger integrato.

WinDriver 6.03
Produttore: Jungo
Su Web www.jungo.com
Prezzo: $1999
Nel CD: WNDRV.EXE

Astrum InstallWizard 1.95.5

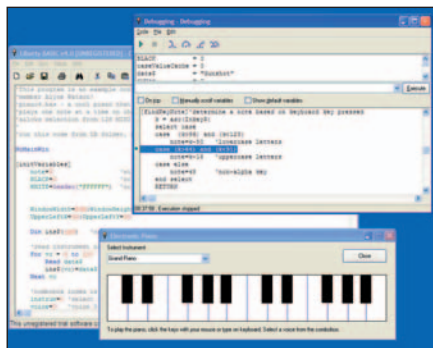
Crea il tuo setup

Astrum InstallWizard è un versatile software per la creazione di pacchetti di installazione. Semplice da utilizzare, grazie all'interfaccia risolta completamente in un wizard. Non rinuncia alla completezza comprendendo il supporto per file JPEG ed MP3.
È possibile utilizzare variabili utente, dividere i file di installazione su più dischi e interagire in vario modo con il registro di Windows. Semplice ed altamente personalizzabile.

Questa nuova versione risolve alcuni bug presenti nella precedente release.

aiw.exe

Liberty BASIC 4.0

Tutta la libertà di cominciare con il piede giusto

Un linguaggio di programmazione sviluppato appositamente per chi si avvicina la mondo della programmazione. Grazie alla funzione *FreeForm* si possono sviluppare complesse interfacce con la semplicità offerta da un ambiente visuale. L'IDE comprende un editor con funzione highlighting sintattico e un completo debugger. Giunto ormai alla quarta versione definitiva, testimoniano la validità dell'ambiente i numerosi sviluppatori che animano le community dedicati a Liberty BASIC sul Web. Versione di prova limitata a trenta giorni.

lb400win.exe

MSDE Manager 1.007

Per gestire database MSDE attraverso una intuitiva interfaccia grafica

Tutte le più comuni operazioni per la gestione di un database MSDE possono essere effettuate per via visuale grazie a MSDE Manager. Sarà possibile aggiungere, modificare ed eliminare qualsiasi elemento: database, tabelle, viste regole, stored procedure, dati degli utenti e tutto quanto è necessario alla vita di una base di dati. E' possibile schedulare le attività più complesse e gestire tutte le fasi di back-up e restore. Versione di valutazione valida quattordici giorni.

msde.exe

OnTime Defect Tracker Windows Edition 3.1

Traccia, gestisce e aiuta a risolvere i bug

Basato su .NET e SQL-Server, OnTime Defect Tracker è un valido aiuto durante lo sviluppo di un progetto software: tutti i bug possono essere tracciati e gestiti attraverso complessi strumenti di ricerca ed analisi. Le informazioni possono ovviamente essere condivise fra tutti i componenti al team di sviluppo, ma sulla base di apposite policy di sicurezza. E' anche disponibile un SDK opzionale che consente di integrare le capacità di defect-tracking all'interno delle applicazioni che sviluppiamo. E' possibile scegliere tra due tipi di interfacce: Windows Client o Web Client. Versione dimostrativa.

OnTimeSetup.msi

Maguma Studio 1.2

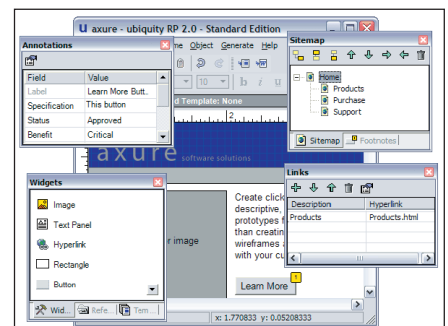
Per creare e gestire siti PHP

Un ottimo ambiente di sviluppo per PHP che, anche grazie ad un eccellente IDE, consente di rendere più piacevole e veloce il "duro" mestiere di creatore e gestore di siti PHP. Sintax Highlighting e debugger integrato sono tra i punti di forza dell'IDE, insieme ad un comodo archivio di snippet che offre già in partenza numerose funzioni ed è completamente personalizzabile. Molto ben fatto il browser per le classi che, attraverso una struttura ad albero, semplifica notevolmente la comprensione della struttura delle applicazioni. Versione di valutazione.

maguma_studio-1.2.0-complete.exe

Ubiquity RP Standard Edition 2.0

Per disegnare l'interfaccia delle tue applicazioni

Un tool per la realizzazione rapida di prototipi per l'interfaccia di applicazioni Web. È possibile simulare il funzionamento dell'interfaccia con l'integrazione di dati fittizi autogenerati e, alla fine del processo, avremo a disposizione delle specifiche complete e di assoluto livello professionale del lavoro svolto.

Versione di prova valida quindici giorni.

UbiquityRP-2.0.5-Standard.exe

Superior SQL Builder 1.3

Per costruire visualmente script SQL

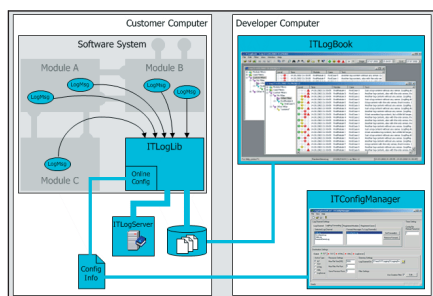
Completamente visuale, consente di costruire non semplici query, ma interi script SQL. Il risparmio di tempo conseguente si deve soprattutto alla riduzione degli errori che è possibile commettere durante la scrittura di lunghi script.
Flessibile e versatile, può essere un valido ausilio a qualsiasi livello, sia agli sviluppatori che agli amministratori di DB. Supporta MySQL, Oracle, Access e SQL Server. Richiede che siano installati il .NET Framework ed il Data Access Components nella versione 2.7 o superiore. Versione dimostrativa valida quattordici giorni:

risulta disabilitato il salvataggio.

superior.zip

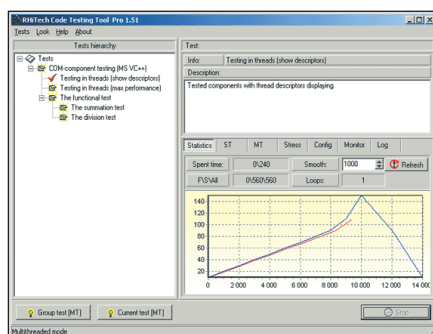
iTech Logging 2.3

Estendi le tue applicazioni con la capacità del logging

Un potente sistema di logging per le nostre applicazioni che si compone essenzialmente di due parti: una libreria integrabile nei nostri progetti ed una plancia di comando per monitorare l'attività delle applicazioni a run-time. La libreria può essere integrata in progetti scritti in un qualsiasi linguaggio di sviluppo (.NET, C++, Java, Delphi, C, progetti COM ed altri ancora), mentre l'interfaccia di gestione semplifica le operazioni di monitoraggio, attraverso un'ampia dotazioni di filtri, che consente di focalizzare l'attenzione di volta in volta sugli aspetti che vogliamo curare. Versione di prova della durata di trenta giorni.

ITLSetup.zip

RHiTech Code Testing Tool Pro 1.51b

Testa le tue applicazioni

Un tool che consente di testare il codice che scriviamo simulando situazioni di reale carico. Le classi possono essere testate sia in ambito multi-thread che single-thread. Gli ambienti supportati sono Delphi e C++ Builder di Borland, ed il Visual C++ di Microsoft. L'installazione include numerosi

esempi. Versione dimostrativa.

RHiTech_CTT.exe

HexToolbox Hex Editor 2.01

Per manipolare file di grandi dimensioni

Un potente editor esadecimale che consente di visualizzare e modificare file di grandi dimensioni (fino a 4 GB). Oltre all'ASCII supporta la codifica EBCDIC tipica dei mainframe. Supporta operazioni di drag&drop e spicca ottime prestazioni nella ricerca raggiungendo il ragguardevole traguardo di 1GB al minuto su un PC di medie prestazioni. Versione di prova valida trenta giorni.

HexToolboxSetup.exe

CDB-Databases Comparator 2.0

Per effettuare il confronto fra struttura e contenuto di due database

Un utile sistema per confrontare due database eterogenei ed, eventualmente, sincronizzarne i contenuti. È previsto il confronto sia dei dati contenuti che della struttura delle basi di dati prese in considerazione. I report sono generati in HTML. Versione di prova valida quattordici giorni.

CDBSetup.exe

Multiple Database Query Analyzer 2.1

Lancia le tue query su qualsiasi DB

Un tool flessibile e leggero che consente di lanciare query su diversi database in parallelo: SQL Server, Access, Oracle, e MySQL. È possibile salvare le query; sono altresì presenti funzionalità di import/export per la migrazione di dati fra database.
L'interfaccia è comoda, anche se lontana da standard professionali, carenza che si riflette anche nella scarsa stabilità complessiva dell'applicazione. Versione di prova valida trenta giorni.

multidbqa.exe

Open Office Developer Build 680_m17

Prova il brivido di personalizzare il tuo office

Continua lo sviluppo della suite open source rivolta alla produttività d'ufficio. OpenOffice si propone come soluzione integrata di applicazioni, multilingue e multipiattaforma, basata sul noto standard XML, utilizzato per lo scambio ed il salvataggio di documenti.
I componenti inclusi sono: un word processor *(Writer)*, un foglio elettronico *(Calc)*, un programma per creare grafici e diagrammi *(Draw)* ed un tool per la creazione di presentazioni multimediali *(Impress)*.
Nella directory trovate anche la completa piattaforma di sviluppo che consente di modificare ed estendere l'applicativo.

OOo_680m17_Win32Intel.zip

Gaim 0.74

Basta un messaggio…

Un instant messenger basato su GTK2 che supporta tutti i più diffusi protocolli, grazie ad appositi moduli integrati.
È possibile connettersi a tutte le piattaforme più utilizzate: AIM, ICQ, Yahoo!, MSN, Jabber, IRC, Napster, Gadu-Gadu e Zephir.

gaim-0.74.exe

Azureus-2.0.4.2

Un client Java per BitTorrent

BitTorrent è una piattaforma open source per il filesharing.
Il successo riscontrato, soprattutto nel mondo Linux, è davvero travolgente.
Presentiamo qui un client Java che si integra in questa piattaforma.

Azureus_2.0.4.2b_setup.exe

Ruby 1.8.0

Il linguaggio Ruby mira ad essere il miglior linguaggio di scripting, a metà fra Perl e Python, offre un pieno supporto alla programmazione object oriented.

Poche regole e veloce ciclo di scrittura-debugging-deploy.
Ruby è un linguaggio interpretato e, molta della sua flessibilità è dovuta al fatto che le variabili non hanno un tipo fisso: non solo non è necessario inizializzarle ma, una volta utilizzate possono cambiare al volo il tipo.

ruby-1.8.0-i386-mswin32.zip

Pathfinding giocando

Gli algoritmi più avanzati di pathfinding si fondano sullo studio di alcuni giochi come gli scacchi. Esaminiamo A* e IDA* entrambi fondati sull'algoritmo *minimax*.

Trovarsi in un intricato labirinto ed avere alle costole esseri che ti vogliono mangiare non è una bella sensazione: bisogna fare alla svelta, farsi venire qualche idea per scoprire una via di fuga che ci salvi la pelle. Ecco cosa si prova nel giocare allo storico Pacman. I patiti di giochi ricorderanno anche Sokoban, in cui un diligentissimo operaio sposta della mobilia (o qualcosa di simile) in ambienti solitamente ingarbugliati ed angusti al fine di non rimanervi intrappolato e sistemare il tutto. Insomma, i giochi fanno un uso massiccio di labirinti o comunque di ambienti da percorrere, dove bisogna trovare il giusto cammino, appunto *pathfinding*. Molti invece, non sanno che è altrettanto vero anche il contrario, ossia che gli algoritmi di ricerca del cammino fanno uso massiccio dei giochi, in particolar modo, quelli di strategia. Nella presente trattazione osserveremo come molti giochi, dal semplice tris agli scacchi, contribuiscono allo studio e lo sviluppo di nuovi algoritmi. Si noterà come solo alcune classi di giochi, i "somma zero" e con informazione perfetta danno un reale contributo allo sviluppo. Investigheremo sul ruolo del famoso algoritmo *minimax* nella ricerca di un cammino ottimo. Implementeremo tanto codice per la risoluzione e l'analisi dei problemi che affronteremo. Non ci resta che concentrarci e tuffarci nel mondo del pathfinding.

COSA C'È DA SAPERE CIRCA A*

Brevemente rivisiteremo uno dei metodi maggiormente usati in relazione al pathfinding. I metodi di ricerca *A** si implementano mantenendo due importanti strutture: le *open list* (lista aperta) e le *closed list* (lista chiusa). Le prime conservano gli stati iniziali (quelli da valutare) mentre le seconde memorizzano gli stati già esaminati (inizialmente è vuota). La soluzione si raggiunge percorrendo un albero di soluzioni parziali che si identificano con i nodi. Le informazioni sui nodi possono trovarsi nelle due liste accennate e sono caratterizzate da due importanti indici euristici, $g(n)$ e $h(n)$. Il primo contiene il costo minimo di quelli conosciuti, mentre $h(n)$ è una stima del costo del particolare nodo (stato). Trovare la soluzione significa trovare il nodo che garantisca migliori performance, ovvero, che presenti il migliore indice $f(n)$ che risulta dalla somma dei due precedenti, $f(n)=g(n)+h(n)$. L'algoritmo *A** mediante un processo interattivo non fa altro che "travasare" stati dalla lista aperta alla lista chiusa valutando ad ogni passaggio il parametro euristico $f(n)$. Si tratta, in altri termini, di verificare costantemente ed in funzioni di mutazioni dell'intera situazione (si pensi ad un ambiente in cui gli ostacoli sono mobili) nuove soluzioni. Si parla di successori del migliore stato: in tal modo l'open list è in continuo cambiamento poiché viene alimentata da nuove soluzioni e svuotata dall'esame di alcune. Inoltre, la lista aperta può accogliere nodi che erano stati estratti e posti nella closed list ma che in seguito alle alterazioni dello stato generale hanno riscontrato diminuzioni di $g(n)$. Il punto cruciale è quindi stimare $h(n)$ correttamente mediante valutazioni euristiche. È chiaro che tale funzione può variare a seconda del tipo di pathfinding che si sta trattando, ad ogni modo deve essere sempre garantita l'ammissibilità del parametro. Tale caratteristica si ha quando non viene mai sovrastimato il costo dello stato, così siamo garantiti circa la generazione della soluzione ottima nel valutare il primo nodo. Per intenderci, nel caso dei comuni campi di applicazione del pathfinding, che sono la ricerca di cammini su aree bidimensionali, $h(n)$ non è altro che la distanza tra il punto corrente n e l'obiettivo da raggiungere. In altri problemi è più difficile da formulare e diventa il vero nodo da sciogliere per poter applicare *A**.

NOTA

GIOCHI SOMMA ZERO

Sono i giochi di strategia come scacchi, go, othello, a due giocatori, per i quali vale la semplice regola di equilibrio per la quale se vince uno l'altro perde e viceversa oppure si patta. Se ad un giocatore è assegnato punteggio 1 se vince 0 se patta e -1 se perde; i giochi a somma zero restituiscono sempre il valore nullo sulla somma dei risultati dei due competitori.

Un tipico gioco (non sarà l'unico che incontreremo come promesso) che viene affrontato con *A** e che è il vero riferimento per le analisi teoriche nella comunità degli studiosi del metodo è il quadrato magico (Fig. 1). Il rompicapo padre del più complesso cubo di Rubick.

***Fig. 1:* Il rompicapo quadrato magico è un esempio di problema che può essere risolto mediante applicazione delle tecniche A*.**

La distanza Manhattan (conosciuta anche come L_1, che è pari alla somma dei valori assoluti delle differenze delle singole componenti) tra le varie tessere è un ammissibile ed effettivo coefficiente euristico che viene usato in *A**.

NOTA

LA PERFETTA INFORMAZIONE

Gli scacchi appartengono alla famiglia dei giochi in cui è fornita perfetta informazione. In ogni momento entrambi gli contendenti hanno tutte le informazioni riguardo allo stato della partita (pezzi e loro posizione). Altri esempi di giochi con perfetta informazione sono la dama, othello e go; mentre non appartengono a tale categoria, ad esempio, il poker (dove la non conoscenza delle carte degli avversari viola le regole della perfetta informazione) o giochi che fanno uso di dadi.

L'ALBERO DELLE SOLUZIONI, OVVERO MINIMAX

Restringiamo la nostra attenzione alla valutazione di algoritmi che implementano giochi strategici con perfetta informazione e che siano somma zero (vedi i box). L'esplorazione delle soluzioni avviene in modo naturale su una struttura ad albero dove ogni nodo può avere più figli. Il pathfinding verrà applicato per trovare il giusto cammino nell'albero (ossia la soluzione di una ben definita casistica di problemi). La generazione di un nuovo stato, una nuova mossa nel caso di giochi a informazione perfetta e somma zero, equivale alla produzione di un nodo figlio. L'albero termina con i nodi foglia che non avendo figli sono le ultime soluzioni da valutare. Tutte le foglie sono raccolte in un vettore che chiameremo in conformità alla natura della struttura, *frutti*. Con riferimento ai giochi tale vettore indica le soluzioni per entrambi i giocatori nella posizione finale.Tra due giocatori, ha più possibilità quello che a soluzioni corrispondenti ha parametri di valutazione positivi, meno l'altro che avrà valore speculare rispetto allo zero, essendo il gioco somma zero. L'applicazione in problemi reali (giochi reali), sebbene siano chiare le linee guida che descrivono l'algoritmo, diventa di notevole complessità implementativa, il che ci costringe a considerare esempi semplici. Il gioco che più si presta ad essere analizzato è sicuramente il tris, conosciuto anche come *naughts and crosses* o anche come *tic-tac-toe*. Ancora ricordo un vecchio film "war game" in cui si sottoponeva il gioco a un elaboratore mostruoso dalle dimensioni eccezionali, che però oggi sarebbe meno potente di un semplice PC. L'analisi del problema si attua mediante albero, come mostrato in Fig. 2. I livelli dell'albero corrispondono alle mosse dei due giocatori; quindi livelli pari per un giocatore dispari per l'altro.

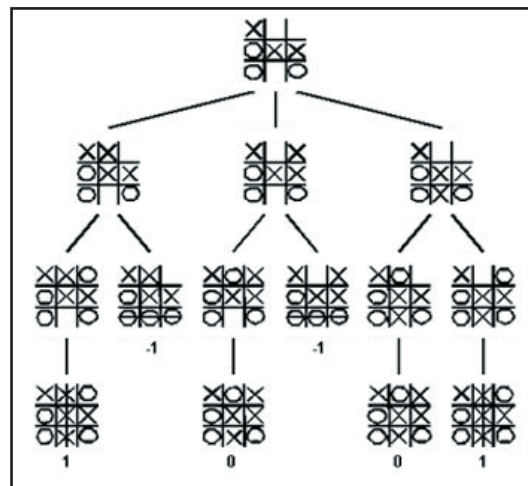



***Fig. 2:* Gioco del tris.**

I due giocatori *X* e *O* saranno indicati rispettivamente come *max* e *min*. La prima mossa mostrata nell'albero, nella radice, è di *X*, mentre, al primo livello immediatamente successivo tocca a *O*. I livelli sono conosciuti come *ply*, così *ply-0* è la radice, *ply-1* il primo livello e così via. Ogni mossa da compiere, corrispondente ad un generico nodo dell'albero: *max* dovrà scegliere quella che massimizza i frutti. Similmente, l'obiettivo di *min* è quello di scegliere un percorso (un nodo figlio) che minimizzi i frutti di *max*, il che implicitamente massimizza i suoi. In funzione del carattere somma zero del gioco si assegna a frutti il valore 1 se vincente, zero se si pareggia e -1 se perdente. Ricordo che per questioni di spazio e poiché si tratta di giochi somma zero è sufficiente mantenere i frutti per uno solo dei due giocatori. Si può quindi assegnare a tutti i nodi dell'albero un coefficiente (un frutto) che altro non è che il valore *minimax*, partendo dalle foglie e risalendo fino alla radice. Rispetto alla Fig. 2 che mostra tali valori per le foglie, in Fig. 3 è riproposto l'albero e il coefficiente *minimax* riportato per ogni nodo.
L'obiettivo di *max,* a partire dalla radice, è quello di individuare un percorso che assicuri o migliori il valore *minimax* della radice. Un tracciato con tali caratteristiche, per uno dei due giocatori, è detto ottimo. Stimare il valore minimax di una posizione significa sviluppare un algoritmo che cerchi in tutto l'albero. La ricerca in profondità fa meno uso della memoria rispetto alla migliore ricerca o della

ricerca in ampiezza. I tre metodi corrispondono alle tre note tecniche di esplorazione degli alberi.

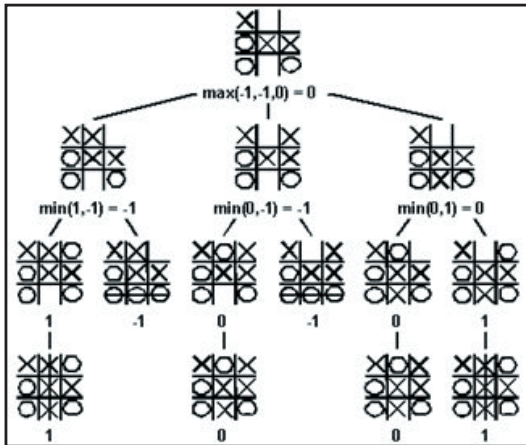



Fig. 3: ***Valori minimax per l'albero associato ad una partita di tris.***

UN PO' DI CODICE

È arrivato il momento di sviluppare del codice che implementi le tecniche solo verbalmente esposte, allo scopo ci aiuterà lo standard dei linguaggi, C++. Le due funzioni riportate di seguito implementano la ricerca in profondità in modo ricorsivo. La funzione *Maximize* assume come input una posizione *p* e fornisce in output un valore *minimax* ottimo, dopo aver opportunamente fatto una ricerca sull'albero.

```
// Implementazione di A*
int Maximize(position p) {
    int numsuccessori;  /* numero totale di mosse */
    int gamma;          /* massimo corrente */
    int i;              /* contatore di mosse */
    int score;     /* punteggio restituito dalla ricerca */
    if (EndOfGame(p)) { return GameValue(p)); }
       gamma = -4;
       numsuccessori = GenerateSuccessors(p);
   for (i=1; i <= numsuccessori; i++) {
        score = Minimize(p.succ[i]);
        gamma = max(gamma, score);
    }
    return (gamma);
}
int Minimize(position p) {
    int numOfSuccessors; /* numero totale di mosse */
    int gamma;              /* massimo corrente */
    int i;                  /* contatore di mosse  */
    int score;    /* punteggio restituito dalla ricerca  */
    if (EndOfGame(p)) { return GameValue(p)); }
       gamma = +4;
       numsuccessori = GenerateSuccessors(p);
    for (i=1; i <= numsuccessori; i++) {
        score = Maximize(p.succ[i]);
        gamma = min(gamma, score);
```

```
    }
    return (gamma);
}
```

Il codice, volutamente, viene lasciato ad un certo livello di astrazione in modo che sia facilmente manipolabile per essere adattato alle diverse applicazioni del metodo. In questa ottica non viene definito il tipo *position* di *p* che indica la posizione corrente e non si implementano le tre funzioni *EndOfGame(position), GameValue(position)* e *GenerateSuccessors(position)*. Ad ogni modo, anche se abbastanza comprensibile dal codice, darò comunque ulteriori precisazioni sul funzionamento delle routine. La prima funzione determina se il gioco è terminato, in tal caso restituisce valore vero; la seconda, accettando una posizione come input, fornisce come *output* il vincitore, o meglio restituisce *minimax* rispetto al giocatore *max;* l'ultima delle funzioni genera un vettore di successori a partire dalla posizione di input, l'output è proprio il numero di tali soluzioni. Le due funzioni *Maximize* e *Minimize* si richiamano reciprocamente in modo ricorsivo. Importante è il ruolo di gamma che mantiene la valutazione della posizione corrente basata su tutte le mosse fatte fino a quel momento. Un miglioramento dell'algoritmo fa in modo che la funzione *maximize* restituisca, oltre che il valore *minimax,* anche la migliore mossa. Volevo ancora una volta sottolineare la flessibilità del codice che si adatta all'implementazione di giochi strategici sviluppando in modo corretto le funzioni sopra citate.

IDA*

Algoritmi più efficienti non effettuano ricerche esaustive su tutto l'albero delle soluzioni, ma effettuano opportune potature di rami nella ricerca. Un sostanziale miglioramento dell'algoritmo *A** è *alfabeta* proposto da uno dei maggiori esperti del campo: Knuth. Tale algoritmo fa uso di funzioni di valutazione che stimano se è il caso di potare alcuni rami qualora le "aspettative" non siano promettenti. Ulteriori miglioramenti di *alfabeta* si fondano comunque sullo stesso criterio di funzionamento e sono *NegaScout [Reinefeld 1983]* e *MTD(f) [Plaat 1996]*. La ricerca nel campo è ancora molto accesa ed i molti tornei tra programmi ne sono una riprova.
L'idea alla base di *alfabeta* è esplorare in profondità l'albero, scendendo di *k* livelli, (solitamente *k* è un valore piccolo), forzando la valutazione dei nodi incontrati. Ad ogni valutazione si può variare il limite *k* di un passo *s*, in modo da aumentare la profondità di analisi (precisamente *k+s*). Nei programmi di scacchi sia *k* che *s* valgono 1, così la

UTILI RIFERIMENTI

Tra gli articoli di soluzioni da me scritti nel corso degli anni ne segnalo alcuni che possono chiarire ed ampliare le tematiche svolte. I numeri da consultare sono: 43, 44 e 45 che affrontano la programmazione del gioco degli scacchi; 39 in cui si affronta il problema del pathfinding in un labirinto con l'uso di backtracking ed in relazione a questo ultimo i due numeri 12 e 21 che puntualizzano l'attenzione rispettivamente sulla pila e la ricorsione.

ricerca avviene per livelli prima *ply-1*, poi *ply-2* e cosi aumentando di 1. La ricerca in profondità così organizzata ha grandi limiti. Ad esempio, se il programma di scacchi che ne fa uso è configurato a tempi fissi (magari bassi) per effettuare mosse, allora nel caso in cui il ramo in profondità che esplora il metodo non porta frutti accettabili, la mossa che si farà potrebbe essere catastrofica. Bisogna quindi mantenere le migliori soluzioni in profondità attuate e, in caso di tempi contingentati, estrarre la migliore ricerca in profondità. Altri algoritmi, invece, individuano un percorso principale e rispetto ad esso tentano nuove ricerche per l'individuazione di nuove soluzioni. Prima di costruire il codice per lo sviluppo di una nuovo metodo concentriamo l'attenzione a programmi di scacchi e ricordiamo che la struttura dati di base dove mantenere i dati è una matrice di trasposizione 8x8 che contiene alcune importanti informazioni della partita (posizione dei pezzi, case attaccate e altro). La funzione riportata di seguito implementa il metodo *IDA** che ha diverse similitudini con l'algoritmo *A** precedentemente trattato. Ad ogni passo, nella generica posizione *p*, si valuta se ci sono dei nodi che possono raggiungere l'obiettivo o se possono essere tagliati immediatamente. Per farlo, si usano valutazioni euristiche tipiche di minimax. Se la profondità *g(p)* più lo stimatore euristico *h(p)* sono maggiori del coefficiente di cut off (limite, è lo spartiacque che fa intraprendere un percorso di ulteriore ricerca piuttosto che di potatura del ramo) indicato con *fmax* si termina la ricerca; altrimenti ricorsivamente si procede verso la ricerca di altre soluzioni.

```
// Algoritmo IDA*
int IDAStar(position p, int g) {
    /* g è il numero di passi per raggiungere la posizione */
    /* fmax è determinato dalle mosse elaborate */
    int numsuccessori; /* numero totale di mosse */
    int done;          /* per capire se è finita la ricerca  */
    int i;             /* contatore di mosse */
    int h;    /* distanza euristica da raggiungere */
    /* Bisogna usare una distanza che sia una stima
                                          ammissibile */
    h = StimatoreDiDistanza(p);
    /* verifica se il nodo è alla portata */
    if (h == 0) { return TRUE; }
    /* valutazione delle funzione euristica */
    if (g+h >= fmax) { return FALSE; }

    /* Potatura, non si possono tagliare tutti i figli del
                                       corrente nodo */
    numsuccessori = GenerateSuccessors(p);
    for (i=1; i <= numsuccessori; i++) {
        done = IDAStar(p.succ[i], g+1);
        if (done == TRUE) { return TRUE; }
    }
    return FALSE;
}
```

La routine *IDAstar(position, int)*, come si può osservare, implementa l'algoritmo precedentemente descritto. La questione adesso si sposta su un interrogativo di semplice formulazione ma di difficile trattamento: come si genera il coefficiente *fmax*? A tale domanda risponde il codice associato alla funzione *ElaboraMosse(position)*. Inizialmente viene assegnato un valore che è proporzionale alla distanza dal punto iniziale e successivamente viene modellato attraverso il richiamo ricorsivo della funzione *IDAstar* mediante la ripetuta correzione del fattore *fmax* dipendente dall'effettivo ritrovamento di un percorso utile (si esamini il ruolo della variabile *bFoundPath*)

```
// Algoritmo che guida IDA*
int ElaboraMosse(position startposition)
{
    int fmax;    /* massima distanza da ricercare */
    int bFoundPath = FALSE;
    fmax = StimatoreDiDistanza(startposition);
    while (bFoundPath == FALSE)
    {
        bFoundPath = IDAStar(startposition);
        if (bFoundPath == FALSE) { fmax += 1; }
    }
    return fmax;
}
```

Studi approfonditi di calcolo numerico hanno confrontato i due metodi esaminati *A** e *IDA**.
Risulta che *IDA** è leggermente più lento ma fa un uso della memoria esponenzialmente minore rispetto a *A**. Inoltre, data la struttura dei dati che tratta è più facile implementarlo con metodi propri della intelligenza artificiale.

CONCLUSIONI

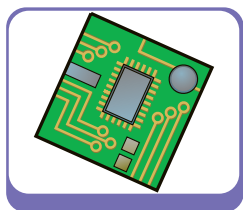
Come abbiamo visto, la valutazione e lo sviluppo di algoritmi di ricerca di soluzioni su alberi, si riconducono a ricerca di cammini e alla produzione di routine che siano in grado di potare rami di albero per rendere la ricerca più veloce senza correre il rischio di tagliare soluzioni interessanti.
Si aprono vasti ed interessanti scenari ai programmatori che intendono analizzare e risolvere problemi con tali metodi. A chi è interessato ad approfondire l'argomento consiglio un puntuale approfondimento delle basi teoriche. Il presente articolo può essere un utile reference per orientarsi nello studio. Presto approfondiremo altri aspetti della questione. Vi aspetto!

Fabio Grimaldi

L'informatica applicata all'elettronica

Una SIM, più operatori

Con questo articolo realizzeremo un nuovo sistema che consentirà, ad ogni possessore di telefono cellulare, l'utilizzo simultaneo di più operatori.

NOTA

Molte Green Card in commercio, hanno come tipo di PIC, il 16F877, totalmente compatibile con il PIC 16F876. Se così fosse, vi consigliamo di cambiare il tipo di PIC nelle impostazioni dell'IcProg, riferendovi a quello in vostro possesso.

Quanti di noi hanno provato ad utilizzare un emulatore? Non dite di no… nessuno vi controllerà il PC in ufficio. Forse l'emulatore più conosciuto e più scaricato, il M.A.M.E., sarà stato installato e provato almeno una volta nel corso dell'utilizzo del nostro beneamato PC.
Ebbene, quello di cui ci accingiamo a parlare in queste pagine, è proprio un emulatore.
"No" direte voi, "Basta, non ne possiamo più", e lo direte con tutte le ragioni di questo mondo. Quello che vi chiedo è di armarvi di un po' di pazienza e leggere queste pagine.
L'emulatore menzionato in queste pagine, non è un emulatore che permette l'esecuzione di vecchi giochi da Bar o di processori per il test di programmi specifici, bensì un emulatore di SIM Card per telefoni cellulari.
Nell'utilizzo del cellulare abbiamo sempre sentito la mancanza, (almeno per quanto mi riguarda) di non poter gestire la telefonata in modo da utilizzare la tariffa più conveniente in quella fascia oraria e con quel tipo di operatore, se non manualmente, costringendoci a servirci di più schede SIM, da sostituire all'occorrenza.
Fino a poco tempo fa, questo era lo "Sport" più praticato nella telefonia mobile. Diversi produttori d'accessori per cellulari, hanno sfoggiato i loro "Gusci" con possibilità di contenere anche 3 SIM Card contemporaneamente. Questo però a discapito, molte volte, dell'estetica del nostro cellulare, e anche per qualche improvviso momento d'empasse dell'elettronica che gestisce l'automatismo dello switch.
Oggi, alcuni operatori si sono aggiornati, dando la possibilità di avere più numeri sulla stessa carta SIM. Handicap di questo sistema è quello di essere costretti ad avere il medesimo operatore per entrambi i numeri. Con queste pagine, aiuteremo il lettore all'utilizzo di un nuovo sistema, che porterà l'utente del telefono cellulare ad utilizzare più numeri di telefono e più SIM, anche d'operatori diversi, switchando da un operatore all'altro semplicemente resettando il telefono e immettendo il nuovo PIN, ovvero riproducendo esattamente la medesima funzione ottenibile tramite un Guscio Multi-SIM.

IL CUORE DEL SISTEMA: L'EMULATORE

L'emulatore che ci permette questa meraviglia è il SIM Emu, creato dalla mente di Daniel Jabif. Molti, forse, già lo conosceranno. Il test che ci accingiamo a realizzare, è stato eseguito con il telefono NOKIA 3650, con il lettore/scrittore SIM: GSM Card Reader della SBS s.r.l. corredato di software Sim Magic, con una Green CARD e con l'accoppiata Lettori/Scrittori per schede con Microcontrollori MultiPIPO/SmartMouse.
È da premettere che l'esecuzione del backup della SIM originale, in alcuni casi, potrebbe danneggiare irreparabilmente la scheda, pertanto, sconsigliamo di compiere tale operazione senza essere "sicuri" del pericolo a cui si va incontro. Inoltre, è necessario che abbiate un po' di dimestichezza con la scrittura dati su di una Gold/ Silver/Green CARD.
Ecco il riassunto dei passaggi da eseguire per inserire, su una sola SIM, due o più operatori:

1. **Programmare una Green CARD trasferen-**

do al suo interno il SIM Emu.

2. **Effettuare un Backup delle nostre SIM Originali da poter far gestire dal SIM Emu.**

3. **Impostare il SIM Emu con i dati ottenuti dal Backup delle SIM Originali.**

Ricordiamo che tutto quanto descritto in queste pagine, è da considerarsi di puro studio e quindi di LIBERO UTILIZZO. Chi deciderà di eseguire il test illustrato in queste pagine, se ne assume tutta la responsabilità. Non è da ritenersi responsabile né l'Autore né la redazione di ioProgrammo qualora se ne facesse uso diverso da quello puramente a TITOLO DI STUDIO.

LA GREEN CARD

Prima di procedere è bene dare uno sguardo allo schema di una Green CARD; al suo interno contiene un PIC (16F876/7) ed una EEPROM (24LC128).
La differenza tra la Silver CARD e la Green CARD è soltanto relativa alla dimensione della EEPROM, infatti la Green CARD con i suoi 128 Kbit, permetterà di contenere fino a 250 numeri in rubrica (ADN), fino a 40 SMS (SMS) e 32 elementi per la lista di selezione fissa (FDN); quelle fornite di PIC ed EEPROM possono essere utilizzate per svariati scopi.

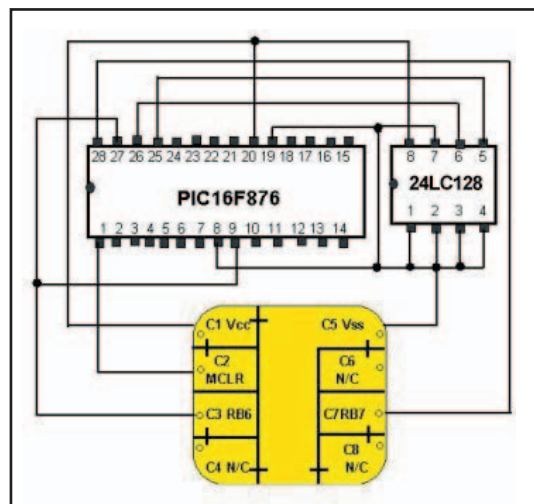



Fig. 1: ***Schema elettrico della Green CARD.***

Un tempo erano utilizzate per la pirateria satellitare ma con l'avvento della nuova piattaforma SKY e del nuovo standard SECA2, questa attività fraudolenta fortunatamente è cessata, almeno per il momento.

Gli utilizzi di una Green CARD non si limitano alla pirateria satellitare né alla clonazione di SIM per cellulare, ma gli usi di studio sono molteplici ed interessanti.
Per avere maggiori dettagli sui Microcontrollori della MicroChip, vi rimandiamo alle pagine Web del costruttore, in cui potrete trovare ampie descrizioni tecniche. In futuro, ioProgrammo, tratterà questo argomento molto più dettagliatamente e tecnicamente.

PROGRAMMIAMO LA GREEN CARD

La scrittura delle schede Green CARD, può avvenire mediante un kit utilizzato in passato per lo studio delle schede satellitari comprensivo di lettore/scrittore di PIC (MultiPipo), Lettore/Scrittore EEPROM (SmartMouse) e di un cavo NULL Modem.
La prima cosa da fare è configurare il file da uploadare nel PIC, in base alla scheda in nostro possesso. Per far questo è necessario eseguire il file SIM_EMU_6.01_CFG.exe, prelevabile direttamente dal sito *http://simemu.cjb.net/* o dal supporto CD-Rom che accompagna la rivista.
Una volta scelto il tipo di scheda e la versione localizzata in italiano scrivendo ITA in Language Suffix, si dovrà impostare il limite massimo di nomi della rubrica e il numero massimo degli SMS. Per una descrizione più approfondita, vi rimandiamo al sito Web dello sviluppatore. Vi consigliamo di impostare i valori che mostrati in Fig.2.

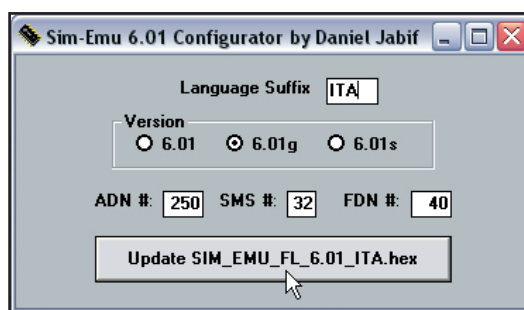
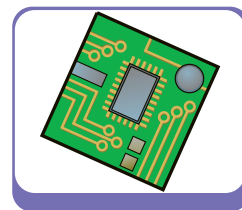



Fig. 2: ***Il pannello di configurazione e selezione del tipo di SimCard.***

Fatto questo, clicchiamo sul pulsante *"Update SIM_EMU_FL_ 6.01_ITA.hex"* che modificherà il file da uploadare nella Green CARD.
Adesso dobbiamo programmare la Green CARD in modo da caricare, al suo interno, l'intero emulatore SimEmu.
Utilizzeremo la versione 6.01 del SIM Emu, ovvero quella compatibile con le "normali"

IMSI
International Mobile Subscriber Identity.

TMSI
Temporary Mobile Subscriber Identity.

Ki
Individual subscribers authentication key
A8
Ciphering key generating algorithm

A3
Authentication algorithm

PIN/PIN2
Personal Identity Number

PUK/PUK2
PIN Unblocking Key.

FDN – Fixed Dialing Numbers list

MCC
Mobile Country Code

Sim Card PhaseII
Sim card di seconda generazione.

NOTA

Il PC dove è stato eseguito il test, è così equipaggiato:

CPU: AMD Athlon XP 2500+

MB: Asus A7N8X Deluxe

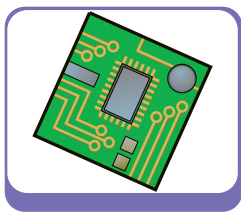
RAM: Corsair Twinx 1GB DDR – 200 Mhz

Green CARD ma anche con le ultimissime Green CARD 2 (16F877/24LC256).
Per quanto concerne il software da utilizzare, consiglio l'ottimo IcProg 1.05A. È possibile trovare su internet nuove versioni del software citato e che potete tranquillamente utilizzare, ma se volete esser certi che tutto funzioni, è conveniente "Allinearvi" alla versione in nostro possesso.
È utile anche sapere che l'accoppiata MultiPIPO/Smartmouse non permette la programmazione della EEPROM se non prima di aver caricato nel PIC un file denominato Loader; tuttavia è bene comunque dire che il software SIM Emu, incorpora già un loader, rendendo superflua l'operazione.
Il Loader, in parole povere, consente la programmazione dell'EEPROM mediante l'utilizzo del PIC, ovvero "costruisce" una sorta di BUS atto a realizzare una comunicazione tra PIC e EEPROM, consentendo la scrittura dell'EEPROM stessa.
La procedura che illustreremo, a solo scopo didattico, prevederà l'utilizzo del Loader. La scelta di utilizzare la coppia MultiPipo/Smart Mouse essenzialmente è dovuta al fatto che tali apparecchi sono molto meno costosi e molto più diffusi, e soprattutto di facile realizzazione.
Procediamo quindi connettendo il MultiPIPO alla porta COM ed eseguendo l'applicazione IcProg.
Il primo utilizzo di IcPRog comporterà, inevitabilmente, il setup di alcune opzioni: scegliamo il tipo di programmatore e la porta su cui lo abbiamo connesso, così come mostrato in Fig.3.

Fig. 3: ***Configurazione iniziale dell'IcProg 1.05A.***

Indirizziamo l'IcProg sul tipo di microcontrollore da utilizzare, nel nostro caso il PIC16F876.
Fatto questo, caricheremo l'opportuno Loader. Dal menu *Opzioni* scegliamo il protocollo "Multimac 214", con frequenza di 3,58 Mhz. Clicchiamo, quindi, sul tasto *OK*.
Non ci resta che inserire la Green CARD nel MultiPIPO e cliccare sul tasto "Programma Tutto".
Se tutto è andato a buon fine, possiamo passare alla scrittura dell'EEPROM: spegniamo il PC e sconnettiamo il MultiPIPO, connettiamo poi lo SmartMouse e riaccendiamo il PC.
Lo spegnimento del PC è importante sia per evitare che vada in corto la Motherboard del nostro PC quando si inserisce il cavo alla porta COM, sia per evitare che si danneggino i chip del kit SmartMouse. Una volta completato il riavvio, è necessario, su alcuni PC, aggiornare l'elenco delle periferiche in Risorse del Computer.
Avviamo nuovamente IcProg e ritorniamo sul pannello principale per impostare come dispositivo, l'M24C128. Siamo ora pronti per trasferire il file *SIM_EMU_EP_6-00.Hex*.
A caricamento ultimato dal menu Settaggi selezioniamo la voce "SmartCard (Phoenix)", inoltre, dal pannello Opzioni, selezioniamo la voce "MultiMac 214" con frequenza "3.58 Mhz". Procediamo quindi con la scrittura dell'EEPROM mediante il tasto "Programma Tutto". Se tutto è andato a buon fine, abbiamo programmato la EEPROM e non ci resta che completare il lavoro.
In questa fase, dobbiamo riprogrammare il PIC inserendoci l'ultimo file *.Hex* modificato in precedenza tramite l'utility *SIM_EMU_6.01_CFG.exe*.
Sconnettiamo lo SmartMouse e riconnettiamo il MultiPIPO. Carichiamo il file *"SIM_EMU_FL_6.01_ITA.hex"* e ricontrolliamo che le opzioni siano impostate come fatto in precedenza per il Loader,
Prima di caricare il file (premendo il bottone "*Programma tutto",* dovrete assicuravi che i bit di configurazione, siano impostati correttamente, in particolare il bit *WRT.*
*G*eneralmente quest'ultimo è impostato automaticamente al termine del caricamento del file, in ogni modo, è sempre opportuno verificate il tutto. Non ci resta che eseguire l'ultima programmazione cliccando sul tasto *"Programma Tutto".*
Una raccomandazione: eseguire, a fine programmazione, la lettura del PIC; in questo modo ci accerteremo che l'intera operazione sia andata a buon fine. La verifica può essere compiuta selezionando dal pannello *Opzioni->Programmazione,* la voce di verifica automatica sia durante che dopo la scrittura del PIC.

Se tutto è stato eseguito correttamente, la nostra Green CARD è stata programmata ed è pronta per essere inserita nel nostro NOKIA 3650.
Prima di iniziare la configurazione vera e propria del Sim EMU tramite il nostro cellulare, è necessario eseguire il backup delle SIM Originali che saranno, successivamente, gestite dall'emulatore.

COME FUNZIONA UNA SIM CARD

La SIM Card, è composta da numerosi campi. Per uno sguardo più approfondito, vi rimandiamo a quanto illustrato in Fig.4. La clonazione avviene mediante attacco di crittoanalisi sull'algoritmo d'encryption COMP128, un particolare A3/A8 che utilizzano i provider. Per capirne bene il funzionamento, dobbiamo dare uno sguardo su come avviene l'accesso alla rete, in questo caso GSM. All'accensione del cellulare, è emessa una richiesta d'accesso alla stazione base che, a sua volta, restituisce un numero casuale con codifica a 128 bit.

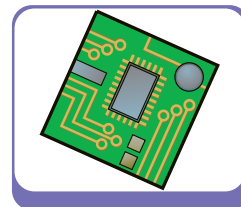
Interno di Sim Card

- *International Mobile Subscriber Identity* (IMSI)
- *Temporary Mobile Subscriber Identity* (TMSI)
- *Individual subscribers authentication key* (Ki)
- *Ciphering key generating algorithm* (A8)
- *Authentication algorithm* (A3)
- *Personal Identity Number* (PIN e PIN2)
- *PIN Unblocking Key* (PUK e PUK2)
- *Rubrica telefonica abbonato*
- *Messaggi SMS abbonato*
- *Lista operatori GSM preferenziali scelti*
- *Campi informativi specifiche GSM Phase 2*

Fig. 4: Contenuto di una Sim Card PhaseII.

Da questo numero (chiamato RAND) ed in base anche al KI, la SIM calcola la risposta (SRES) usando l'algoritmo A3. La stazione base recupera anch'essa, dall'archivio del gestore, il codice KI associato all'utente. In seguito, tramite il suo KI e al RAND, ne calcola l'SRES. Se il risultato coincide con quello inviato dal cellulare, l'identificazione è andata a buon fine.
È proprio questo che faremo con il nostro PC: interrogare la SIM. Avvieremo quindi la procedura d'autenticazione fino a che sia restituita una risposta dopo l'applicazione del COMP128 alla sua chiave. Questo fa sì che, una volta analizzata la risposta, sia possibile ricavare il valore della chiave segreta. Rimane il problema di "bucare" il sistema, ovvero interrogare la SIM più volte (query), utilizzando particolari algoritmi che permettono l'apprendimento del codice d'accesso. Inoltre, dopo svariati studi, è stato notato che il chip A5, disponendo di una chiave a 64bit, ne utilizza soltanto 54.
Ai restanti 10 bit viene attribuito il valore "0x000000".
Questo tipo d'attacco di crittoanalisi richiederà alcune ore dato che saranno eseguite, come spiegato, delle interrogazioni in sequenza per migliaia di volte, e questo è proprio l'algoritmo utilizzato dal SIM Scan.

CLONARE LA SIM CARD ORIGINALE

Partiamo con una premessa: l'unica scheda a non poter essere clonata è quella dell'operatore Wind.
Tale scheda è totalmente cifrata in tutte le sue versioni e quindi non è immediato risalire ai dati in essa contenuti. L'operazione di clonazione funziona egregiamente in tutte le schede TIM, dalle prime 64K in giù, a tutte le SIM Vodafone Omnitel 16.1K. Per le Vodafone Omnitel 16.2K in su, e le sim@ctiva, non è possibile risalire ai dati e quindi effettuare una copia di sicurezza efficiente. È possibile tentare ugualmente la lettura dei dati, senza compromettere in alcun modo l'integrità della SIM originale.
È altresì possibile procedere con la clonazione delle prime schede dell'ormai "defunto" operatore BLU.
La copia di backup delle SIM richiede diverse ore, quindi se non si ha un computer dotato di uno degli ultimi processori in circolazione, l'operazione potrebbe scoraggiare anche il più paziente degli utenti.
La clonazione che abbiamo eseguito di una carta SIM della TIM, ha richiesto 3 ore e 34 min. di lavoro, utilizzando una postazione PC abbastanza all'avanguardia.
Per una completa descrizione del PC utilizzato nel test, si rimanda al box informativo presente in queste pagine. La clonazione avviene tramite un particolare software denominato *SIM Scan*, disponibile sia nel supporto CD-Rom allegato alla rivista, sia sul web (*www.ioprogrammo.it*, sezione download).

NOTA

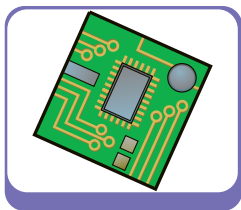
Quando tagliate la SIM prestate attenzione a non piegare il bordo della protezione conduttiva del PIC/EEPROM. È importante fare molta attenzione dato che potreste rendere non più utilizzabile la Green CARD.

Bisogna sapere che la clonazione non è una copia vera e propria della SIM, ma è una “copia” alcuni campi cardine che dovremo immettere nelle impostazioni dell’emulatore.
I campi principali che andranno estratti sono l’IMSI e il KI. Per clonare la nostra SIM Card, dobbiamo avvalerci del succitato SIM Scan e di un lettore/scrittore di SIM Card.

Fig. 5: ***L’inserimento della SIM Card nel lettore della SBS. Lo trovi su www.sbs-power.com, acquistalo online per 39,30 euro.***

NOTA

E’ consigliabile utilizzare il seguente metodo che facilita il compito di ricordare sia il PIN2 che il PUK, e cioè quello di associarli alla posizione. Supponiamo di aver scelto come posizione della configurazione, la numero 7. Potremmo utilizzare questi PIN2 e PUK: 7777 e 77777777. Facilissimi da ricordare per la posizione a cui sono associati.

ella nostra prova, abbiamo utilizzato l’hardware SBS GSM Card Reader, tale lettore è corredato da software per la gestione della rubrica telefonica, degli SMS e delle suonerie non polifoniche.
Il SIM Scan si occuperà di tutte le interrogazioni e dell’estrapolazione di campi superflui. Inseriamo la nostra SIM Originale nel lettore della SBS, prestando attenzione ad introdurre la SIM dopo aver collegato il lettore alla porta COM del PC. Inoltre fate attenzione al verso e alla direzione d’inserimento della vostra SIM Card. In caso di inserimento errato, non otterrete la connessione del lettore. Potete riferirvi a quanto illustrato in Fig. 5 per orientarvi e per evitare inutili arrabbiature.
Inserita la SIM Card, possiamo eseguire il SIM Scan
Come già detto in precedenza, questa è un’operazione che richiede da 1 a 12 ore di lavoro. Ricordiamo, ancora una volta, che in caso di fallimento potrebbe essere danneggiata irrimediabilmente la vostra SIM.
Inserita la Sim, sarà necessario effettuare la connessione; un MessageBox indicherà che l’operazione è stata compiuta con successo: non ci resta che avviare la procedura di Backup. Ultimato il backup sarà necessario annotare una serie di dati che porteranno alla “creazione” del KI.
Ricordiamoci anche di annotare il valore IMSI. La ricostruzione del KI è realizzata secondo lo schema proposto in Fig. 6.
La lettura deve avvenire per colonna partendo dalla prima colonna, che nella Fig.6 ha come suo primo valore “16”, e segnarli in modo orizzontale fino alla riga 8. Successivamente, continuare a segnarli in orizzontale, ripartendo dalla colonna 2, contrassegnata dal valore “14”, fino a riga 8. Otterrete un KI del tipo:

*16 ** ** ** ** ** ** ** 14 ** ** ** ** ** ** **.*

Per scoraggiare qualsiasi apprendista “furbo”, teniamo a precisare che anche i primi valori, mostrati in Fig.6, sono di fantasia.
Ora che abbiamo i dati, possiamo finalmente passare alla configurazione del SIM Emu tramite il nostro cellulare.

SIM Pro Backup

SIM Backup | Connect | Help | Exit

KEY Code.	VALUE	Elapse
1	16 14	00:25:53
2	[illegible]	00:31:20
3	[illegible]	00:58:41
4	[illegible]	00:12:36
5	[illegible]	00:55:01
6	[illegible]	00:10:05
7	[illegible]	00:20:55
8	[illegible]	00:00:00
		03:34:32
IMSI	[illegible]	[illegible]

SIM Connected... | 17:04:35 | 7-002A00

Fig. 6: ***L’insieme dei dati estrapolati dalla nostra SIM Card originale.***

NOTA

Attenzione: non tenete attivate contemporaneamente, la SIM Emulata e la SIM Originale perché il gestore, accorgendosi dell’anomalia, bloccherebbe la SIM ed eventualmente il Cellulare.

CONFIGURIAMO IL SIM-EMU

Eccoci arrivati alla configurazione dell’emulatore. Un avviso prima di tutto: le Green CARD, quasi sempre, sono vendute in un package plastico che necessita d’essere inciso per poter essere inserito nel cellulare. Purtroppo quest’operazione è veramente la più delicata di tutto il test. Vogliamo consigliarvi di lasciar perdere la precisione maniacale ma di badare, quando serve tagliarla, alle misure che consentono al cellulare almeno il corretto blocco della scheda. Per esempio se osserviamo un Nokia 3310, possiamo notare che l’importante è tagliare con precisione soltanto la larghezza più che la lunghezza. Questo in parte

vale anche per il Nokia 3650 ma potrebbe non valere per altri tipi di cellulari. Ricordiamo che a detta dell'autore, il SIM Emu è stato testato sulle seguenti marche di cellulari:

- Nokia
- Siemens
- Alcatel
- Philips
- Ericsson
- Motorola
- Maxon
- Panasonic
- Mitsubishi
- Nec
- Sagem
- Samsung

Prima di inserire la Green CARD, dovrete impostare la lingua del telefono in "Italiano" e non su "Automatica" per esser certi che sia forzata quella Italiana e non quella Spagnola, utilizzata dall'autore del software in oggetto. Inoltre l'autore ha preventivamente riempito le prime 3 posizioni con gestori spagnoli per evitare problemi con telefoni Bloccati. Le posizioni inserite dall'autore, sono state impostate con i seguenti PIN: 0000, 1111, 2222; utilizzeremo uno di questi all'accensione del cellulare, successivamente potranno essere cambiati secondo le proprie esigenze. Entrati nella schermata principale, clicchiamo sul tasto "Menù" e selezioniamo l'icona contrassegnata dal nome "SIM". Possiamo subito notare la presenza di un'icona con il nome "Sim-Emu…", cliccandoci sopra avvieremo l'applicazione.

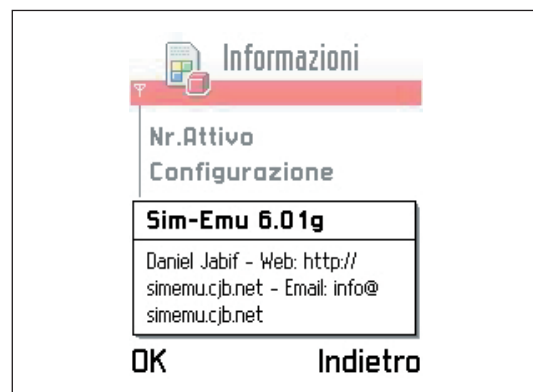



Fig. 7: Come si presenta il Sim Emu.

Configuriamo la prima delle 8 posizioni disponibili che conterranno le SIM emulate: dirigiamoci sulla voce "Configura" e selezionamola, sarà richiesto il PIN2 ovvero il valore reimpostato "1234" (anche questo modificabile successivamente). La successiva operazione sarà quella di scegliere la posizione da programmare. Per far questo, selezioniamo la voce "Posizione" e digitiamo un numero compreso tra 3 e 7. Fatto anche questo, verrà chiesto di inserire il codice "IMSI" e successivamente il "KI", codici annotati in precedenza (vedere paragrafo Clonare la Sim Card Originale).

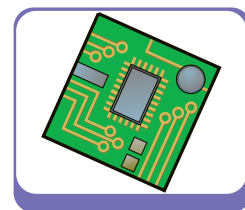
IMSI = CC / MNC / MSIN
CC - Country Code
MNC - Mobile Network Code
MSIN - Mobile Station Identification Number.

Fig. 8: Shema codice IMSI.

È da tener presente che l'emulatore compierà un test sulla validità dell'IMSI immesso, nel caso in cui il test dovesse fallire, in seguito non sarà più richiesto il KI. Cosa diversa se invece l'IMSI viene validato. Verrà infatti salvato e visualizzata la maschera di inserimento del codice KI. Se il codice KI inserito sarà esatto, sarà richiesto il codice PUK da associare alla posizione e successivamente, anche il codice PIN. Adesso non rimane altro che selezionare la voce, "Cambi Nr.Tel.", scegliere la posizione appena creata e aspettare il riavvio del telefono. Una volta avviato, verrà richiesto il PIN, inseriremo quello precedentemente scelto nella creazione della nuova posizione. Se il tutto è andato a buon fine, basterà attendere e la connessione alla base e… la SIM Clonata sarà in funzione. Esiste ora un'unica accortezza prima di procedere all'utilizzo: configurare il numero di SMS contenuti dalla SIM ed il numero delle voci in rubrica. Per la Green CARD il massimo per gli SMS è 40, mentre le voci in rubrica arrivano ad un massimo di 250.
Avrete anche la possibilità di creare una descrizione della posizione. Per un maggior dettaglio sulle varie voci, che per ragioni di spazio non abbiamo potuto qui elencare, vi rimandiamo alla lettura della pagina ufficiale dell'emulatore.
Ora siamo pronti per giocare con il nostro emulatore… ops… pronti per utilizzare la nostra SIM Emulata!

CONCLUSIONI

Spero che queste poche pagine, aguzzino l'ingegno, inoltre mi auguro che ci siano suggerimenti e nuove idee su questo mondo così affascinante.

Marcello Scala

NOTA

Se doveste avere problemi, potrebbe essere utile configuare l'IcProg in Direct I/O invece che mediante Windows API)

SUL WEB

http://simemu.cjb.net/
sito ufficiale dell'Emulatore.

http://www.sbs-power.it
(Produttore del lerrore SIM Card Reader)

http://www.microchip.com

http://www.InfinityUSB.com

http://users.net.yu/~dejan/
(SIM Scan 2.0)

http://www.ic-prog.com/
(Ic Prog)

Prossimo passo: "Internet di oggetti"

Nei primi anni novanta, una piccola notizia su Wired annunciava la nascita di una misteriosa applicazione sviluppata dall'univeristà dell'Illinois. L'applicazione si chiamava Mosaic.

di Steve Meloan

Nel giro di pochi anni, un intero comparto industriale è stato costruito attorno a Mosaic e ai suoi successori. In realtà Mosaic non è stata la prima applicazione di quel tipo, fu la prima applicazione a rendere semplice l'attività. Un'altra tecnologia, altrettanto potenzialmente rivoluzionaria, è di recente emersa al simposio inaugurale dell'EPC (Electronic Product Code), tenutosi a Chicago lo scorso settembre: la Electronic Product Code Network, una infrastruttura tecnologica aperta, messa a punto da un consorzio di aziende e sviluppatori. La rete EPC, utilizzando i compatti tag RFID (Radio Frequency ID), consentirà ai computer di riconoscere automaticamente l'identità di ogni oggetto di uso quotidiano, monitorandone la posizione, programmando eventi e compiendo azioni su questi oggetti. Questa tecnologia consentirà di creare una vera e propria "Internet di oggetti". La tecnologia RFID avrà un grande impatto in molti settori: dal manifatturiero ai trasporti, dalla medicina alle scienze, dal settore farmaceutico alla pubblica amministrazione.

EVOLUZIONE O RIVOLUZIONE?

Molti osservatori concordano nel ritenere che la prossima ondata di crescita nel settore tecnologico sarà al di fuori del mercato dei PC. Ci troviamo agli albori di un'epoca che vedrà la nascita di oggetti e dispositivi interconnessi che ancora sono ancora inimmaginabili. In realtà, già ora esistono alcune eccitanti applicazioni che allargano lo spettro tecnologico. Ad esempio, NASCAR.com offre un cruscotto virtuale, basato su applet, che visualizza in tempo reale la telemetria delle vetture durante una gara, registrando posizione, velocità, giri del motore, frenate, insieme a molti altri dati. Nel frattempo, la NASA, con l'aiuto di GE Medical, è ora in grado di monitorare in tempo reale i dati biologici degli astronauti dello Space Shuttle (come pressione, respirazione e frequenza cardiaca). Anche gli alberi sono parte del network: cinquanta sensori sono stati collocati su altrettante sequoie del Wisconsin dall'università di Berkley, per monitorarne lo stato di salute. I sensori registrano la luce, l'umidità e la temperatura, consentendo agli scienziati di controllare il microclima che circonda ogni singolo albero. L'estrema diversificazione di questi dispositivi interconnessi offre una concreta dimostrazione della legge di Metcalfe.

Robert Metcalfe, uno degli sviluppatori di Ethernet, postulò che l'utilità di una rete cresce con il quadrato del numero di nodi connessi alla rete stessa. (Immaginate un po' quanto sarebbe utile una rete telefonica con due soli apparecchi connessi!). "Nel futuro, qualsiasi oggetto di una qualche importanza sarà connesso, in un modo o in un altro" ha affermato John Fowler, Software CTO di Sun Microsystems. "Ed una volta che sono connessi, possiamo raccogliere dati dai diversi dispositivi e comunicarli ad altrettanti dispositivi, creando nuove reti di conoscenza." È esattamente ciò cui stiamo assistendo oggigiorno. "Molte persone pensano ai PC e ai PDA quando pensano ad oggetti connessi alla rete", continua Fowler "ma ormai stiamo connettendo alberi, macchine da corsa e astronauti alla rete. E lo spettro va via via allargandosi".

Fig. 1: Una formica gioca con dei chip RFID.

RFID

Con il rilascio ufficiale dell'Electronic Product Code Network, stiamo per vedere il paradigma "Internet di oggetti" entrare nel suo momento di gloria: il mondo del consumo di massa. Guardando il fenomeno nel suo complesso, il termine Auto-ID fa riferimento a qualsiasi classe di tecnologie di identificazione utilizzate nel commercio per aumentare l'automazione, ridurre gli errori e migliorare l'efficienza.

Fig. 2: Struttura di un tag RFID.

Queste tecnologie includono: codici a barre, smart card, sensori, riconoscimento vocale e tutta la biometrica. Attualmente, la tecnologia Auto-ID maggiormente sotto i riflettori è l'RFID *(Radio Frequency Identification)*. RFID è una tecnologia che prevede l'utilizzo di trasmettitori wireless miniaturizzati per etichettare i singoli oggetti, identificandoli univocamente. Le etichette (tag) RFID consentono alle aziende di tracciare automaticamente gli oggetti, generare eventi ed effettuare delle azioni sugli oggetti stessi. I chip RFID hanno attualmente raggiunto la dimensione di 0,3 millimetri (la punta di una matita) e sono disponibili

in diverse tipologie: attivi (dotati di batteria), semi passivi (sempre con batteria) e passivi (alimentati dall'energia indotta dei lettori di tag). Ogni tipologia presenta vantaggi, svantaggi e caratteristiche diverse.

TAG RFID (RADIO FREQUENCY IDENTIFICATION)

Un tag è composto da un chip RFID e da una antenna. Esistono tre tipi diversi di tag: attivi, semi-passivi e passivi. Ognuno si presta ad utilizzi diversi, così come è indicato nella tabella successiva.
Le tecnologie per i transponder RFID (trasmettitori/risponditori) sono già da tempo operative: dai badge utilizzati in molte aziende per l'accesso ad aree riservate, al telepass utilizzato per il pedaggio autostradale anche in Italia. I tag RFID passivi hanno in genere un raggio d'azione limitato a pochi metri: il campo elettromagnetico emesso dai lettori induce una corrente nella etichetta, permettendole di trasmettere una risposta wireless a bassa energia. Tutte queste tecnologie si basanosu architetture proprietarie, senza mai rispettare uno standard. Al fine di ottenere una rete di oggetti veramente utile, è necessario assicurare l'adozione di standard aperti e l'interoperabilità globale.

L'AUTO-ID CENTER

Fondato nel 1999, l'Auto-ID Center è una organizzazione di ricerca non-profit che riunisce 103 fra istituzioni e aziende, che assieme rappresentano un fatturato di oltre 1500 miliardi dollari). Il centro ha il suo quartiere generare al MIT ed una serie di laboratori sparsi per le più prestigiose università del mondo. Gli sponsor fondatori sono giganti del calibro di Uniform Code Council, EAN International, Gillette e Proctor & Gamble. Gli attuali membri annoverano nomi altrettanto notevoli: Coca-Cola, Home Depot, Johnson & Johnson, Kraft, Lowes, PepsiCo, Pfizer, Sara Lee, Target, Unilever, UPS, e Wal-Mart. Al fine di realizzare la visione proposta dall'Auto-ID Center, è necessario innanzitutto definite, costruire, testare e distribuire una infrastruttura aperta e globale che si poggi su Internet. Il che significa: hardware economico, protocolli, e linguaggi di sviluppo. Sun Microsystems, grazie alla sua ventennale esperienza nel campo delle tecnologie cross-platform, è certamente qualificata per giocare un ruolo chiave nei lavori dell'Auto-ID Center. Già dal 2000, Sun ha preso parte al Technology Board ed al Software Action Group dell'Auto-ID Center. I due obiettivi prioritari del Centro sono stati abbassare i costi (sia delle etichette RFID che dei lettori), e definire l'infrastruttura tecnologica necessaria per utilizzare questi oggetti in un ambito globale e conforme alla rete EPC per le industrie.

CODICE EPC: UN MIGLIORAMENTO DI UPC

Al centro della infrastruttura per RFID appena rilasciata dall'Auto-ID Center c'è il codice EPC, dati numerici trasmessi dai tag. In realtà, il codice EPC ha l'obiettivo di costituire la prossima generazione dell'UPC *(Universal Product Code):* i codici a barre che si trovano ormai su pressoché tutti i prodotti di largo consumo. A differenza dell'UPC, l'EPC è stato ideato per operare non solo senza bisogno di un contatto visivo, ma anche per identificare il singolo oggetto. Per promuovere l'adozione della tecnologia RFID, in particolar modo per l'etichettatura dei singoli oggetti, la riduzione di costo è un fattore essenziale. Il prezzo di ogni etichetta è attualmente sui 0,15 euro, ma si aspetta a breve un crollo del prezzo a 4 centesimi di euro. Sul fronte lettori, il prezzo di aggira fra i 1500 e i 2500 euro, ed anche qui la caduta prevista è forte: 80 euro. Standard aperti, adozione globale e concorrenza fra più produttori: tutto ciò servirà ad abbassare rapidamente i prezzi. Con in mente il problema-costi, l'Auto-ID Center ha definito più versioni delle specifiche EPC (64 bit, 96 bit, e 256 bit). La Fig. 4 mostra il dettaglio della specifica a 96 bit che consente di identificare univocamente 268 milioni di aziende. Ogni azienda può avere fino a 16 milioni di classi di oggetti distinte, con 68 miliardi di numeri seriali per distinguere i singoli oggetti per ogni categoria. A differenza dei codici a barre UPC, le specifiche EPC forniscono dunque una identificazione univoca del singolo oggetto:

- **L'intestazione (8 bit) specifica la versione di EPC**
- **Il campo manager (28 bit) fornisce il nome dell'azienda**
- **il campo object class specifica la classe del prodotto**
- **il campo numero seriale identifica univocamente l'oggetto.**

	Tag attivo	Tag semi-passivo	Tag passivo
Sorgente di energia	Batterie integrata nel tag	Batteria integrata, utilizzata solo per le operazioni del chip. Le comunicazioni avvengono con l'energia irradiata dal reader	Tutta l'energia, sia per il chip che per le operazioni, è fornita dalle onde radio del reader
Disponiblità del segnale del tag	Sempre acceso, 30 metri	Solo all'interno del campo del reader	Solo all'interno del campo del reader, e comunque inferiore a 3 metri
Potenza del segnale del tag	Alta	Bassa	Molto bassa
Potenza richiesta al reader	Molto bassa	Bassa	Molto alta
Applicazioni tipiche	Utile per tracciare oggetti di valore che hanno bisogno di essere tracciati su lunghi percorsi.		Utile per merci prodotte in ampi volumi, e nei casi in cui è possibile avere reader e oggetto vicini

TABELLA 1: Le diverse tipologie di tag RFID

***Fig. 4:* Schema delle specifiche EPC a 96-bit.**

L'EFFETTO WAL-MART

Le nuove tecnologie vanno e vengono: alcune non riescono mai a raggiungere una reale popolarità. RFID ed EPC sembrano essere diverse. Basta un'occhiata alla lista delle compagnie che aderiscono all'Auto-ID Center per avere un'idea di quanto va-

sto sia l'interesse verso queste nuove tecno logie. Uno sguardo a cosa è successo in passato, può aiutarci a leggere l'immediato futuro. Il brevetto originale per i codici a barre fu depositato nel 1952. Ci vollero venti anni perché fosse approvato uno standard ma, ancora nel 1982, solo 15.000 fornitori utilizzavano i codici a barre. Ebbene, nel 1987, c'erano già 75.000 fornitori che avevano adottato i codici a barre. Cosa era successo? Semplicemente, nel 1984 Wal-Mart, la più grande catena commerciale degli Stati Uniti, aveva cominciato ad utilizzare questa tecnologia. Tornando al presente, Wal-Mart ha recentemente fatto sapere ai suoi 100 principali fornitori che, a partire dal primo gennaio 2005, devono dotare tutti i loro prodotti di tag RFID. I restanti 12.000 fornitori hanno tempo fino al 2006. Quando Wal Mart parla, l'industria sta ad ascoltare. Sembra chiaro che il tempo per la tecnologia RFID è ormai arrivato.

APPROFONDENDO

Gli standard chiave e le tecnologie attualmente definite dall'Auto-ID Center e dai suoi partner includono:

- *EPC*
- *Tag RFID*
- *Lettori di tag*
- *Savant servers*
- *Object Naming Service (ONS)*
- *Product Markup Language (PML)*
- *EPC Information Service (EPC IS)*

Object Naming Service

Modellato sulla base *Domain Name System (DNS)* di Internet, l'*Object Naming Service (ONS)* consente di accertare efficacemente l'identità degli oggetti dotati di tag RFID, mappando il codice EPC sulle informazioni associate presenti su un server PML.

Product Markup Language

Il Product Markup Language (PML) è un linguaggio XML-based, appositamente creato per fornire uno standard universale per la descrizione di oggetti fisici all'interno di una rete EPC. PML è utilizzato per immagazzinare informazioni come: la composizione di un prodotto, numero di lotto, data di produzione, modo di utilizzo, modalità di conservazione ed altro ancora. Utilizzando le informazioni immagazzinate in PML e la rete EPC, un rivenditore può aggiornare le informazioni nel momento stesso in cui effettua l'ordine, può settare dei trigger che abbassano il prezzo all'avvicinarsi della scadenza di un prodotto, ritirare i prodotti scaduti e molto altro ancora.

```
<AutoIdEvent>
   <EventId>AAAABBBBCCCCDDDD000011112002110613O434</EventId>
   <SensorId>AAAABBBBCCCCDDDD00001111</SensorId>
   <TagId>0000111122223333444445555</TagId>
   <TagId>0000111122223333444445554</TagId>
   <Timestamp>2002-11-06T13:04:34-06:00</Timestamp>
</AutoIdEvent>
```

Fig. 6: Un esempio di dati in formato PML.

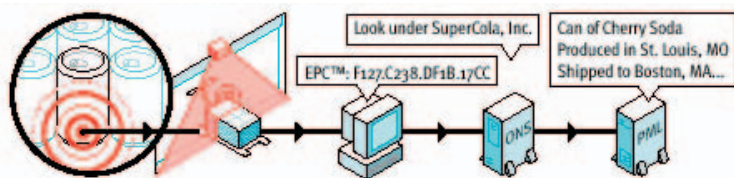
Con il nuovo Network EPC, fabbricanti, distributori e rivenditori potranno tracciare la posizione delle merci lungo tutta la catena di distribuzione.

1. Un codice EPC è impresso all'interno di un microscopico "smart tag" e attaccato ad un oggetto. Con una dimensione di 400 micron quadrati, i tag sono più piccoli di un granello di sabbia. Questi tag consentono di tracciare gli oggetti in modo completamente automatico.
2. I lettori RFID possono riconoscere ogni singolo tag ed inviare i dati relativi ad un PC su cui giti Savant.
3. Il middleware Savant si poggia a sua volta su un database *ONS (Object Name Service).*
4. L'ONS rimappa il codice EPC verso un URL in cui è sono immagazzinate (in formato PML) tutte le informazioni relative agli oggetti.
5. Il server PML contiene informazioni sul singolo oggetto, sul produttore, sulle modalità di vendita insieme ad altri dati ancora.

Fig. 5: Come lavora un network EPC.

L'architettura Savant

Come il nome lascia intuire, l'elemento Savant svolge un ruolo particolare nell'infrastruttura EPC. Con la mole di dati che, verosimilmente, dovranno essere gestiti in alcuni scenari commerciali in cui i lettori RFID potrebbero dover riconoscere centinaia di oggetti al secondo, potrebbero andare in crisi quelli che sono i limiti dell'infrastruttura che consente la trasmissione dei dati stessi. È dunque necessario porre uno strato che faccia da filtro fra i lettori di tag e le rete dell'azienda. Savant si pone all'estremo margine della rete, fornendo degli adattatori per i lettori che consentono l'interfacciamento verso diversi fornitori hardware, evitando l'invio di dati duplicati, facendosi carico di un parte delle regole di processo imposte dalla logica business e smistando i dati all'interno di tutta l'infrastruttura.

LA TECNOLOGIA SUN

Sun Microsystems è stata la prima azienda a postulare che "La Rete è il Computer" e parte dunque in pole position nella rivoluzione di Auto-ID. In una "Internet di oggetti", gli sforzi di Sun per la sicurezza, la scalabilità la compatibilità cross-platform tra sistemi di rete diversi, non sono mai stati così importanti. Con J2EE, J2SE, J2ME, e JavaCard, Sun può offrire un spettro completo di soluzioni cross-platform. Il 16 settembre 2003, Sun ha annunciato la sua nuova iniziativa in ottica Auto-ID. Julie Sarbacker è a capo della nuova Auto-ID Business Unit all'interno di Sun che lavorerà con partner e clienti per sviluppare e rilasciare soluzioni Auto-ID/EPC. La soluzione architetturale proposta da Sun include il Sun EPC Information Server (essenzialmente il Sun ONE Integration Server con dei componenti applicativi Auto-ID) e il Sun EPC Event Manager (gioca il ruolo di Savant), che sono in fase di sviluppo e saranno rilasciati nel corso del 2004.

LA MASSA CRITICA

L'utilizzo di RFID nell'industria appare vicino a raggiungere la massa critica. Oltre alla già menzionata strategia pianificata da Wal Mart che partirà nel 2005, Gillette ha reso noto di aver acquistato 500.000 tag RFID, grazie ai quali sperano di ridurre i casi di merce non disponibile, ridurre i costi di lavorazione, ridurre i furti ed i fenomeni di contraffazione. Steve David, CIO di Procter & Gamble, ha affermato che l'azienda conta di risparmiare 1,5 miliardi di dollari sui costi della catena produttiva at-

traverso l'adozione di Auto-ID. La Michelin, che produce 800,000 pneumatici al giorno, sta prendendo in considerazione l'inserimento di tag RFIS in ogni gomma prodotta. La Delta Airlines, sta testando RFID su un numero limitato di voli, etichettando 40,000 valige per ridurre il numero di smarrimenti e migliorare il processo di smistamento. L'esercito americano ha sistemato tag su 270,000 tra container e veicoli, riuscendo a tracciarne la dislocazione attraverso 40 paesi. Inoltre, Michael W. Wynne (segretario della Difesa degli Stati Uniti) ha annunciato un ambizioso piano (sul modello Wal-Mart) che prevede l'adozione dei tag RFID da parte di tutti i fornitori delle forze armate statunitensi a partire dal gennaio 2005. La Visa sta analizzando la possibilità di inserire tag RFID nelle sue smart card, in modo che sia possibile portare a termine le transazioni senza che il cliente debba aprire il portafogli. La Banca Centrale Europea sta addirittura considerando l'opportunità di annegare tag RFID individuali all'interno delle banconote, per ridurre i fenomeni di contraffazione e di riciclaggio di denaro sporco. Inoltre, attraverso i tag, le banche potrebbero contare grosse quantità di banconote in pochi secondi. Ovviamene i problemi che potrebbero sorgere sono numerosi e non riguardano solo le libertà civili dell'individuo che sarebbero pesantemente danneggiate dall'ultimo brandello ancora disponibile di commercio anonimo: un malvivente potrebbe avvantaggiarsi della cosa, sapendo in anticipo quanti soldi una persona si trova nelle tasche. Con queste problematiche ancora da risolvere, risulta chiara la necessità di una tecnologia sicura nel dominio Auto-ID. I vari attori ed i dispositivi presenti all'interno di un network Auto-ID devono essere verificati e affidabili anche se le specifiche proposte dall'Auto-ID Center non sono rivolte ad un ambito di sicurezza. Ancora una volta, le tecnologia Java si presenta come soluzione ideale, essendo stata pensata sin dall'origine avendo ben presente le problematiche di sicurezza di rete.

Fig. 8: L'infrastruttura EPC di Sun.

NUOVI MODI DI PENSARE

Per andare in contro ai bisogni ed alle sfide della nascente "Internet di oggetti", gli sviluppatori avranno bisogno di cambiare le loro nozioni su cosa sia un dispositivo connesso e su come possa essere utilizzato per fornire servizi interessanti e vantaggiosi. In poche parole, è tempo di cominciare a pensare in termini diversi dalla vecchia logica PC-Palmari. John Fowler di Sun quantifica questa evoluzioni in termini di due "megatrend". "Il primo megatrend è da individuare nella riduzione dei costi che ha portato con se la possibilità di connettere alla rete praticamente qualsiasi cosa. Il secondo megatrend è che la ubiquità con cui è possibile accedere agli oggetti cambierà enormemente il modo in cui le persone si relazioneranno con gli oggetti stessi. In qualità di sviluppatore di applicazioni, è dunque doveroso fermarsi un attimo a riflettere e chiedersi: il fatto che tutto è connesso alla rete, come cambierà il modo in cui io processo le informazioni?" Un perfetto esempio di questo nuovo modo di pensare lo possiamo trovare nell'etichettatura dei prodotti farmaceutici. L'utilizzo più ovvio sarebbe limitato al controllo dell'inventario ma, gli sviluppatori più pronti a cogliere le nuove opportunità hanno già immaginato una applicazione per gli ospedali che prevede un tag per ogni paziente e per ogni trattamento rendendo ambedue parte integrante della rete. Un lettore di tag RFID riconoscerebbe entrambi gli attori, consentendo la somministrazione dei trattamenti solo ai pazienti corretti.

Fig. 7: Architettura EPC Savant.

GUARDANDO AVANTI

La rapida evoluzione dell'Auto-ID vede nuovi sviluppi praticamente ogni mese. Nel settembre del 2003, l'Auto-ID Center ha presentato la versione 1.0 delle specifiche. Questi standard sono ora nella fase di ratifica e, benché non siano fissate delle scadenze, i documenti ufficiale delle specifiche dovrebbero essere già rilasciati al momento in cui leggerete questo articolo.
Nel frattempo, l'Auto-ID Center ha esaurito il suo compito e, dal 26 ottobre 2003, il Centro ha ufficialmente terminato i suoi lavori. Tutto lo sviluppo tecnologico è stato trasferito ad EPCglobal (*www.epcglobalinc.org*) che si occuperà della futura crescita degli standard. EPCglobal è una società non-profit formata da EAN International e dalla Uniform Code Council. Sarà proprio EPCglobal a distribuire i codici alle aziende che ne faranno richiesta.

traduzione a cura di
Raffaele del Monaco

SUL WEB

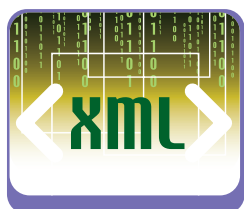
www.autoidlabs.org
wwws.sun.com/software/solutions/auto_id
www.epcglobalinc.org
www.uc-council.org/

Il nuovo anello di congiunzione tra Java e XML

JAXB nuova modalità di interazione

JAXB è la nuova soluzione offerta da Sun per il dialogo fra Java e XML. Oltre a descrivere questa tecnologia, mostreremo un'applicazione che legge e scrive la propria configurazione in un file XML.

WEB SERVICES DEVELOPER PACK

È un pacchetto contenente le tecnologie utili per costruire Web Services con la piattaforma Java 2. Esso raggruppa diverse componenti quali ad esempio JAXB, JAXP, Tomcat,ecc.

Le tecnologie Java e XML agevolano il lavoro degli sviluppatori nello scambio di informazioni e di applicazioni sulla rete. Finora l'anello di congiunzione tra queste due tecnologie è stato rappresentato dal JAXP, consistente in un insieme di specifiche per l'elaborazione di documenti XML, definendo così due possibili metodi di parsing: il DOM ed il SAX. La nuova soluzione offerta da Sun è il JAXB (Java Architecture for XML Binding), una nuova tecnologia Java che permette di generare classi Java a partire da schemi XML. Tale soluzione permette di accedere ad un documento in maniera più semplificata e trasparente dal punto di vista del codice e di ottenere una più efficiente gestione della memoria rispetto alla tradizionale metodologia DOM. L'applicazione che illustreremo e che farà uso di tale nuova tecnologia, si occupa della lettura e della scrittura della propria configurazione in un file XML. La necessità di tale funzionalità è molto frequente, in quanto l'utilizzo di un file di configurazione per una qualsiasi applicazione, piccola o grande che sia, è spesso necessaria ed inoltre il formato XML si rivela particolarmente efficace. Mediante il JAXB, vedremo come, nel caso del nostro esempio, la gestione dei parametri di configurazione si riveli particolarmente veloce ed immediata senza che il programmatore debba intervenire attivamente nei processi di lettura e di scrittura del documento XML. Esattamente come per il JAXP, anche il JAXB consiste in un insieme di specifiche ed una Reference Implementation delle API è disponibile all'interno del pacchetto di sviluppo di Web Services (Java Web Services Developer Pack). Nell'ultima versione, la 1.1, del Java WSDP, è presente l'implementazione delle specifiche JAXB 1.02. Il cuore del JAXB è il binding compiler, un compilatore che ha lo scopo di generare un insieme di classi Java a partire da uno schema XML (come riportato in Fig. 1). Grazie alle classi generate ed alle classi di supporto del JAXB, è quindi possibile:

- **passare da un documento XML ad un insieme di oggetti Java (e viceversa). Tali funzionalità sono note come unmarshalling e marshalling.**
- **eseguire la validazione di un gruppo di oggetti Java rispetto ad uno schema XML.**

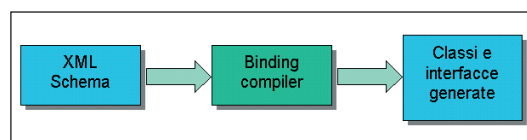



Fig. 1: Processo di binding per la generazione di classi ed interfacce.

L'accesso ad un documento XML, mediante JAXB, da parte di un'applicazione Java si compone di due passi. Il primo passo consiste nel riportare lo schema del documento in un gruppo di classi che rappresenta il modello di dati dello schema stesso. Tale passo viene realizzato dal binding compiler introdotto in precedenza. Nel pacchetto di sviluppo di Web Services è fornito il tool di binding denominato *xjc.bat* (o *xjc.sh* per Unix), la cui sintassi è la seguente:

```
xjc [-options] schema
```

Il parametro *schema* indica il file dello schema XML di partenza. Tra le possibili opzioni c'è la possibilità di specificare la directory di output, il package delle classi generate, ecc. L'output consisterà in un insieme di interfacce e di classi che le implementano. Queste ultime sono raggruppate nel package *impl* (figlio di quello definito come target nelle opzioni) e sono dipendenti dalla Reference Implementation utilizzata (pertanto classi generate con JAXB non sono utilizzabili con una implementazione diversa).
I metodi di accesso agli attributi sono i classici *get* e *set* tipici dei javabeans. Passiamo ora a vedere il secondo passo, denominato *unmarshalling*, che consiste nel leggere il documento XML e creare una struttura di oggetti che corrispondono alle istanze

delle classi generate al primo step. Tale gruppo di oggetti, essendo dipendente dallo schema XML, risulta più efficiente in termini di memoria rispetto all'insieme di oggetti basati su DOM (che viceversa è più generale). L'implementazione JAXB offre, oltre al binding compiler, anche le API necessarie per effettuare le operazioni di marshalling, unmarshalling e validazione. Queste sono contenute nel package *java.xml.bind*.

ACCESSO AD UN FILE XML

In un articolo precedente abbiamo realizzato ciò mostrando un pattern basato su SAX. Stavolta applicheremo lo stesso esempio utilizzando pertanto le nuove funzionalità offerte dal JAXB. Partiamo da un file di esempio, denominato *cfg1. xml*, e riportato di seguito. Esso utilizza uno schema contenuto nel file *cfg.xsd*, disponibile insieme ai sorgenti dell'articolo.

```
<!--La configurazione dell'applicazione.-->
<server-config xmlns:xsi=
        "http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="cfg.xsd">
    <!—Specifica la configurazione di accesso al
                                        database.-->
    <db-access db-driver=
                "oracle.jdbc.driver.OracleDriver" db-url=
                "jdbc:oracle:thin:@kyoto:1521:ORADB"
            user="admin" password="jack" />
    <!—Specifica la configurazione del logging.-->
    <logging debug="true" file="./out.txt"/>
    <!—Specifica il path del file di policy.-->
    <java-security policy-file="./java.policy"/>
    <!—Specifica la porta utilizzata e se il registro RMI
                deve essere creato all'interno del processo
                            dell'applicazione o meno.-->
    <rmi-registry external="false" port="1099"/>
  </server-config>
```

Sono presenti pochi elementi:

- **db-access, relativo alle proprietà di accesso al database (driver da utilizzare, URL del database ed utenza e password).**
- **logging, per le informazioni sul logging in modalità di debug su file.**
- **java-security per impostare il file di policy da utilizzare .**
- **rmi-registry per i settaggi del registro RMI (numero della porta e se utilizzare o meno un registro esterno al processo Java del server).**

Ora passeremo a mostrare in dettaglio come eseguire le operazioni di lettura e scrittura nel caso di un file XML come quello illustrato. A tale scopo, utilizzeremo una classe, denominata *ConfigObject*, la quale sarà responsabile di offrire, mediante i suoi metodi, tali operazioni alla nostra applicazione.
Il primo passo, come visto precedentemente, è quello di generare le classi Java a partire dallo schema. Per far ciò, basta lanciare il seguente comando

```
xjc –p config cfg.xsd
```

dove abbiamo indicato che le classi e le interfacce generate dovranno essere contenute nel package config. In Fig. 2 è mostrato il class diagram con le interfacce e le classi generate (possiamo notare che queste ultime si trovano nel sottopackage denominato *impl*). Osserviamo che, per ogni elemento definito nello schema, viene generata un'interfaccia il cui nome è quello dell'elemento stesso. Ad esempio, per l'elemento *db-access* otteniamo l'omonima interfaccia *DbAccess*, la quale estende *javax.xml.bind.Element* (che non ha metodi in quanto agisce solo come indicatore del tipo di dato) e da *DbAccessType*. Per capire il perché dell'esistenza di quest'ultima interfaccia, osserviamo la porzione di schema che definisce l'elemento *db-access*.

```
<xs:element name="db-access">
  <xs:complexType>
    <xs:attribute name="db-driver" type="xs:string"
                                      use="required"/>
    <xs:attribute name="db-url" type="xs:string"
                                      use="required"/>
    <xs:attribute name="user" type="xs:string"
                                      use="required"/>
    <xs:attribute name="password" type="xs:string"
                                      use="required"/>
  </xs:complexType>
</xs:element>
```

In realtà tale elemento viene definito a partire da un tipo di dato complesso *(complexType)* che risulta essere un gruppo di attributi. Pertanto, il binder com-

Fig. 2: *Class diagram contenente il risultato del processo di binding.*

NOTA

XML SCHEMA

Rappresentano una diversa soluzione al problema della definizione della struttura e dei tipi di un documento (un'altra soluzione è data dai DTD, document-type definitions). Gli schema sono più potenti e versatili in quanto permettono di descrivere relazioni strutturali e tipi di dati non definibili con i DTD. Il linguaggio utilizzato per gli schema è denominato W3C XML Schema Language.

piler genera, per tale tipo di dato, un'interfaccia denominata *DbAccessType* in quanto il nome del tipo non è specificato. Tale interfaccia, definendo i metodi *get* e *set* per gli attributi, agisce quindi da padre per la *DbAccess* che invece si riferisce all'elemento *db-access*. Non entriamo ulteriormente in dettaglio ma in ogni caso per maggiori informazioni sul processo di binding si può fare riferimento alle risorse indicate a fine articolo. Per effettuare il processo di unmarshalling del documento (riportato in Fig. 3), aggiungiamo alla classe *ConfigObject* un metodo read, il quale per prima cosa crea un oggetto *JAXBContext*, contenuto in *javax.xml.bind*. Questa classe rappresenta il punto di partenza per l'accesso alle API del JAXB e riceve nel suo costruttore uno o più nomi di package che contengono le interfacce generate dal binding compiler. Nel nostro caso gli passiamo il nome del package *config*.

Fig. 3: **Nel processo di unmarshalling, il risultato è un gruppo di oggetti contenenti i dati presenti nel documento XML.**

```
public void read(String fileName) throws JAXBException {
  JAXBContext jc = JAXBContext.newInstance("config");
```

Quindi si crea un oggetto *Unmarshaller*, il quale controlla, mediante il suo metodo *unmarshal*, il processo di lettura del documento. Grazie al metodo *setValidating* possiamo decidere se attivare o meno la validazione del documento XML rispetto allo schema definito.

```
Unmarshaller unmarshaller = jc.createUnmarshaller();
    unmarshaller.setValidating(true);
```

Ora procediamo con la lettura del documento, specificando il nome del file XML da leggere. Il metodo ritorna un oggetto *ServerConfig* che corrisponde all'elemento più esterno definito nello schema.

```
    root = (ServerConfig) unmarshaller.unmarshal(
                              new File(fileName));
```

La struttura ad albero, definita dagli elementi del documento XML, è esattamente riprodotta in una analoga, contenente però le corrispondenti istanze delle classi generate, ed in cui l'oggetto *root* ne è la radice. Pertanto possiamo accedere alla configurazione del logging mediante il seguente frammento di codice (della classe *Application*), in cui leggiamo il file *cfg1.xml*, otteniamo il riferimento all'oggetto *ServerConfig* e da esso passiamo all'istanza del *LoggingType*.

```
  ConfigObject.getInstance().read("cfg1.xml");
ServerConfig config =
               ConfigObject.getInstance().getRoot();
LoggingType logging = config.getLogging();
  System.out.println(logging.getFile());
  ...
```

Un'osservazione va fatta relativamente al metodo *unmarshal*. La classe *Unmarshaller* ne offre differenti versioni con le quali è possibile leggere documenti XML, specificando anche altre sorgenti, diverse da un file, quali ad esempio un oggetto *InputStream*, un URL o un nodo *DOM*.

COSTRUIRE UN DOCUMENTO XML

Finora la nostra applicazione legge la sua configurazione da un file XML creando un insieme di oggetti in memoria (specificati dalle classi generate) e leggendone le proprietà. In mancanza di tale file, però, l'applicazione non potrebbe essere eseguita. In tal caso, come miglioramento, possiamo prevedere che essa parta, utilizzando una configurazione di default creata in memoria, e che successivamente la salvi in un file. Ci troviamo pertanto nella situazione del marshalling (illustrato in Fig. 4), ossia nel dover costruire un documento XML a partire da un insieme di oggetti. Vedremo quindi come ciò sia possibile mediante il JAXB. In generale, per la costruzione di un documento XML, si può utilizzare il solo DOM, ma non il SAX e ciò perché quest'ultimo non permette di effettuare nessuna operazione di manipolazione dei dati in memoria, operazione che invece si rende necessaria in quanto occorre popolare il contenuto del documento in memoria. Lo svantag-

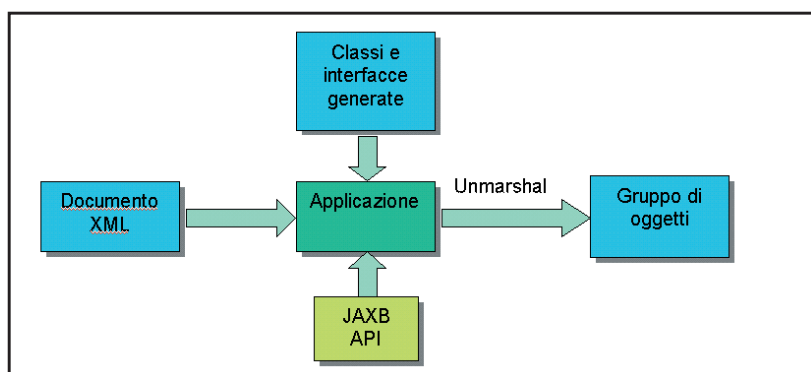



Fig. 4: **Nel processo di marshalling, i dati presenti in un gruppo di oggetti vengono resi persistenti all'interno di un documento XML.**

BINDING DI DEFAULT E SUA ESTENSIONE

Le specifiche JAXB descrivono il comportamento di default per il mapping da un insieme di componenti di uno schema XML ad un gruppo di componenti Java. In ogni caso è possibile costruire uno schema in cui definire delle dichiarazioni che sovrascrivono o estendono tale comportamento.

gio del DOM, però, è quello di dover conoscere la struttura ad albero del documento. In tal caso, occorre creare un parser che costruisca tale albero e successivamente, utilizzando i metodi DOM, navigare attraverso gli oggetti dell'albero che contengono i dati di cui si ha bisogno. L'approccio mediante il JAXB è viceversa più diretto e si basa sui seguenti passi:

- **binding dello schema, ossia generazione delle classi mediante il binder compiler. Tale operazione è identica a quanto visto precedentemente per la lettura del documento XML.**
- **creazione della struttura ad albero.**
- **marshalling della struttura.**

Iniziamo dal secondo passo. La struttura ad albero contenente gli oggetti relativi ai dati del futuro documento XML può essere creata o mediante unmarshalling (come visto) o mediante la classe *ObjectFactory,* la quale è generata durante il processo di binding dello schema. Utilizziamo, come esempio, il metodo *loadDefault* della classe *ConfigObject,* il quale crea gli oggetti relativi alla configurazione di default per la nostra applicazione.

```
public void loadDefault() throws JAXBException {
ObjectFactory objFactory = new ObjectFactory();
    ServerConfig config = objFactory.createServerConfig();
    LoggingType logging =
                    objFactory.createLoggingType();
    logging.setFile("./out.txt");
    logging.setDebug(true);
    config.setLogging(logging);
    ...
```

A questo punto, passiamo al terzo passo, il marshalling della struttura, realizzata tramite il metodo *write* della classe *ConfigObject.* Come nel caso di prima, creiamo un'istanza *JAXBContext* specificando il package delle classi di binding. Quindi costruiamo un oggetto *Marshaller,* il quale si fa carico, mediante il metodo marshal, del processo di scrittura del documento. Il metodo *setProperty* permette di settare alcune proprietà. In questo caso diciamo al *Marshaller* di formattare i dati XML con ritorni a capo ed identazione.

```
  public void write(String fileName) throws
             JAXBException, FileNotFoundException {
JAXBContext jc = JAXBContext.newInstance("config");
Marshaller marshaller = jc.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_
            FORMATTED_OUTPUT,new Boolean(true));
```

Infine invochiamo il metodo *marshal* specificando l'oggetto che contiene il root della struttura ad albero ed il target di uscita.

```
    marshaller.marshal(config,new
                    FileOutputStream(fileName));
```

Anche in questo caso, come nel processo di unmarshalling, si possono indicare anche altri target, come ad esempio un oggetto *OutputStream* o un nodo *DOM.*

AGGIORNARE UN DOCUMENTO XML

Un ulteriore miglioramento da apportare alla nostra applicazione di gestione della configurazione potrebbe essere quello di consentire all'utente di modificare tale configurazione in modo interattivo, mediante ad esempio una serie di finestre grafiche, piuttosto che agire sul file XML. In questo caso, l'applicazione deve lavorare direttamente con gli oggetti della struttura ad albero esistente in memoria (creata con valori di default o letta da un file XML), modificandone le proprietà.
Successivamente la configurazione in memoria verrà salvata mediante il processo di marshalling illustrato in precedenza.

```
  ConfigObject.getInstance().read("cfg1.xml");
  ServerConfig config = ConfigObject.getInstance()
                                        .getRoot();
  logging = config.getLogging();
  ...
  logging.setFile("./new_out.txt");
ConfigObject.getInstance().write("cfg3.xml");
```

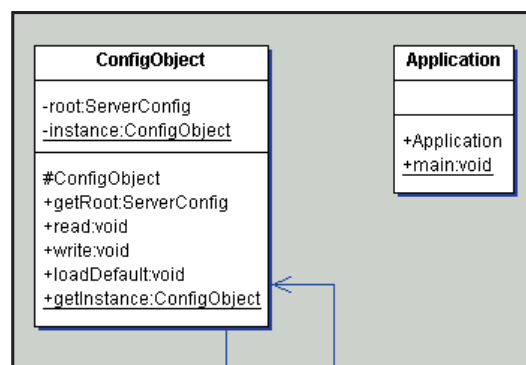



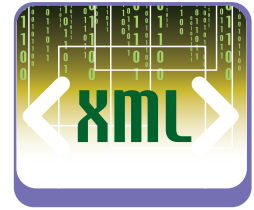
Fig. 5: ***Class diagram contenente le due semplici classi dell'applicazione di esempio.***

In Fig. 5 è mostrato il class diagram della piccola applicazione di esempio che abbiamo creato.

CONCLUSIONI

Abbiamo così visto la semplicità e velocità di utilizzo della tecnologia JAXB applicata ad un esempio reale di applicazione Java.

David Visicchio

BIBLIOGRAFIA

- **THE JAVA ARCHITECTURE FOR XML BINDING (JAXB) – FINAL V1.0 Gennaio 2003 Sun Microsystems, Inc**
- **JAVA WEB SERVICES DEVELOPER PACK V1.1** http://java.sun.com/webservices/webservicespack.html
- **W3C XML SCHEMA LANGUAGE** http://www.w3.org/TR/xmlschema-0

L'AUTORE

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

Un client di posta multipiattaforma

Un Outlook in Java

parte seconda

Prosegue il tutorial dedicato allo sviluppo di un client di posta elettronica scritto con Java. Questo mese saranno passate in rassegna le funzionalità delle API JavaMail dedicate al POP3.

Un client di posta elettronica deve fornire diverse funzionalità di base. Nel corso della prima parte di questa tutorial abbiamo scoperto come sia facile, con le API JavaMail, gestire l'invio di un messaggio di posta elettronica, attraverso un server SMTP. In questa seconda parte esamineremo l'operazione diametralmente opposta, vale a dire il recupero della posta elettronica, sfruttando le API JavaMail per l'impiego di un comune servizio POP3.

COME RICEVERE LA POSTA ELETTRONICA

Cominciamo da un semplice programma per riga di comando, capace di connettersi ad una casella di posta elettronica remota e di mostrare la lista dei messaggi ricevuti:

```
import java.util.Properties;
import java.util.Date;
import javax.mail.*;
public class RiceviMail {
   public static void main(String[] args) throws
                                          Exception {
      // Il server POP3 da impiegare per la ricezione.
      String pop3 = "pop3.miohost.com";
      // I dati per il login.
      String username = "pippo";
      String password = "ciao";
      // Creo un oggetto Properties vuoto.
      Properties props = new Properties();
      // Ottengo la sessione JavaMail.
      Session session = Session.getDefaultInstance(
                                          props, null);
      // Ottengo un oggetto Store per il recupero della posta.
      Store store = session.getStore("pop3");
      // Connetto l'oggetto Store al server remoto.
      store.connect(pop3, username, password);
      // Accedo alla cartella della posta in arrivo.
      Folder inbox = store.getFolder("INBOX");
      inbox.open(Folder.READ_ONLY);
      // Ottengo la lista dei messaggi conservati.
      Message[] messages = inbox.getMessages();
      // Passo in rassegna l'elenco della posta.
      for (int i = 0; i < messages.length; i++) {
         System.out.println(
            "Messaggio " + i + ": " +
            messages[i].getFrom()[0] + "\t" +
            messages[i].getSubject() ); }
      // Chiudo la cartella della posta in arrivo.
      inbox.close(false);
      // Chiudo la connessione con il server remoto.
      store.close();
      // Avviso l'utente del termine del programma.
      System.out.println("Operazione completata."); }
}
```

Prima di compilare ed eseguire il programma, naturalmente, modificate il contenuto delle tre stringhe dichiarate in apertura, in modo da sfruttare una vostra casella di posta elettronica per il test. Proprio come nel caso esaminato nella prima parte del tutorial, tutto comincia dal recupero di un'istanza della classe *javax.mail.Session*. La sessione di JavaMail recuperata in questa occasione non deve fare utilizzo di particolari proprietà del sistema, come invece avviene nel caso dell'invio, e pertanto gli è stato fornito come argomento un oggetto *Properties* vuoto. Subito dopo aver recuperato la sessione d'utilizzo, incontriamo un nuovo tipo di oggetto: *javax.mail.Store*. *Store*, letteralmente, vuol dire deposito, e le API JavaMail utilizzano gli oggetti *Store* per astrarre le caselle di posta elettronica remote. Connettere un oggetto *Store*, di fatto, vuol dire collegarsi al server POP3 corrispondente ed iniziare ad interagire con esso per il recupero e la gestione delle missive elet-

troniche ricevute. Poiché il POP3 non è l'unico protocollo fruibile per il recupero della posta elettronica, ogni volta che si ottiene uno *Store* è necessario specificare il tipo di comunicazione desiderata, come argomento del metodo *getStore()* di *Session*:

```
Store store = session.getStore("pop3");
```

Una volta che si è connessi al servizio remoto, è possibile accedere al proprio elenco di posta elettronica. Anche in questo caso, le API JavaMail offrono alcune astrazioni che semplificano la programmazione di un client di posta elettronica. Ci si trova davanti alla possibilità di vedere la posta ricevuta immediatamente suddivisa in più cartelle, proprio come avviene con i più comuni client di posta. La posta in arrivo sarà contenuta nella cartella denominata *INBOX*. Per il protocollo POP3, *INBOX* è inoltre l'unica cartella disponibile. Se si utilizza un altro protocollo di ricezione, come IMAP (che è supportato dalle API JavaMail), si avranno ulteriori possibilità. Ciascuna cartella remota è rappresentata da un differente oggetto *javax.mail.Folder*. L'oggetto *Folder* di volta in volta desiderato può essere ottenuto invocando il metodo *getFolder()* di *Store:*

```
Folder inbox = store.getFolder("INBOX");
```

Una volta ottenuto un appiglio verso la cartella desiderata, è possibile entrare al suo interno richiamando il metodo *open()* di *Folder*. L'accesso può avvenire in sola lettura (*Folder.READ_ONLY*), oppure in lettura e scrittura (*Folder.READ_WRITE*):

```
inbox.open(Folder.READ_ONLY);
```

A questo punto è finalmente possibile accedere alla lista dei messaggi conservati all'interno della cartella aperta. Il metodo necessario è *getMessages():*

```
Message[] messages = inbox.getMessages();
```

Si ottiene così un array di oggetti *javax.mail.Message*. Già abbiamo incontrato la classe *Message*, nella sua derivazione *MimeMessage*, nel corso della prima parte di questo tutorial. Naturalmente, ciascun oggetto *Message* ricevuto astrae e contiene un differente messaggio presente all'interno della cartella esplorata. I metodi messi a disposizione da *Message* (controllate pure nella documentazione delle API JavaMail) permettono l'accesso alle differenti parti del messaggio. Una volta completate le operazioni di recupero e lettura, è necessario chiudere la cartella utilizzata:

```
inbox.close(false);
```

L'argomento fornito al metodo *close()* di *Folder* specifica il comportamento che l'oggetto deve tenere nei confronti di quelle missive che, durante l'utilizzo, sono state contrassegnate come messaggi da cancellare. Se l'argomento è *true*, i messaggi verranno realmente eliminati. Sulla cancellazione delle missive ricevute torneremo nel corso dei prossimi appuntamenti. Prima di abbandonare il programma, bisogna chiudere anche l'oggetto *Store* ancora connesso:

```
store.close();
```

Eseguendo il software otterrete un risultato simile a quello mostrato in Fig. 1.

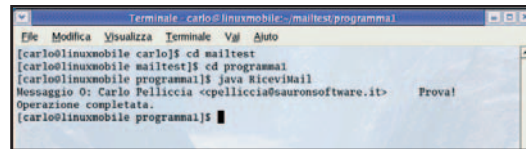



Fig. 1: ***L'applicazione RiceviMail mostra sulla riga di comando l'elenco dei messaggi ricevuti.***

UN SOFTWARE PER LA RICEZIONE

Ora che abbiamo chiarito i concetti fondamentali per la ricezione della posta elettronica, diventa possibile realizzare un piccolo client con interfaccia grafica (il codice completo lo trovate sul Web, in *MailRecever.java*:

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.mail.*;
public class MailReceiver extends JFrame {
  ...
  // Punto di ingresso del programma.
  public static void main(String[] args) {
    MailReceiver mailReceiver = new MailReceiver();
    mailReceiver.show();}
}
```

Di nuovo, nella classe *MailReceiver*, c'è davvero poco. Tutto il codice mostrato nell'esempio del paragrafo precedente è stato "riciclato" all'interno di una interfaccia grafica Swing. Adesso i dati per la connessione e la ricezione della posta sono a discrezione dell'utente. L'unica reale novità del codice consiste nella gestione delle eccezioni che è stata associata alle operazioni di recupero. Le API JavaMail, infatti, dispongono di un set di eccezioni per amministrare ogni possibile anomalia. Il loro impiego è tutt'altro che difficile: come al solito, è sufficiente consultare la documentazione delle API per scoprire a cosa corrisponda ogni possibile eccezione.

PREREQUISITI RICHIESTI

La realizzazione di quanto è mostrato in questo tutorial richiede, da parte del lettore, alcuni prerequisiti di base:

- **discreta conoscenza del linguaggio Java e della sua piattaforma;**
- **discreta conoscenza dei concetti di base del networking;**
- **discreta conoscenza dei meccanismi tipici di un servizio di posta elettronica.**

I lettori più attenti avranno già osservato due cose:

1) Nel metodo *showSelectedMessage()* si fa riferimento ad una classe, *MessageViewer*, che non fa parte né delle API JavaMail né del codice dell'esempio.
2) Al termine della ricezione della posta, nel metodo *run()* di *ReceiveDialog*, gli oggetti *inbox* e *store* non vengono chiusi con i rispettivi metodi *close()*, come la prassi richiederebbe.

Per quanto riguarda il primo punto, *MessageViewer* è una classe che tra pochissimo andremo a realizzare. Per risolvere il secondo dubbio, invece, è necessario comprendere meglio il funzionamento delle API JavaMail nelle fasi di ricezione. Nel momento in cui accediamo ad una cartella remota (ad esempio proprio *INBOX*) e chiediamo l'elenco dei messaggi in essa conservati, non scarichiamo in un solo colpo tutto il contenuto di ciascuna missiva. Per rendere più snelle le operazioni di rete, e per non intasare la memoria locale, la chiamata a *getMessages()* di un oggetto *Folder* ci restituisce solo le informazioni generali dei singoli messaggi presenti nella cartella. Il completamento del download avverrà soltanto quando andremo a richiedere esplicitamente il contenuto di ogni singolo messaggio. Il software che stiamo sviluppando mostra, nella sua finestra principale, solo le intestazioni della posta ricevuta. Il contenuto sarà scaricato e mostrato proprio da quella classe *MessageViewer* che tra poco realizzeremo. Se la cartella *INBOX* e lo *Store* che rappresenta la connessione fossero chiusi, non sarebbe più possibile eseguire il completamento del download, o almeno sarebbe necessario connettersi di nuovo. Poiché siamo in una fase di studio iniziale, lasceremo i due oggetti aperti, in modo da poter completare il download di ogni singolo messaggio senza incappare in ulteriori problemi. Chiarito ciò, andiamo a scrivere il codice di *MessageViewer*:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.mail.*;
class MessageViewer extends JFrame {
  // Costruttore.
  public MessageViewer(Message m) throws Exception {
    // Richiamo il costruttore della classe base.
    super("Messaggio");
    // Assemblo la GUI.
    JTextArea text1 = new JTextArea(
                                    (String)m.getContent());
    text1.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(text1);
    scrollPane.setPreferredSize(new Dimension(300, 200));
    JPanel north = new JPanel(new GridLayout(2, 1, 3, 3));
    north.add(new JLabel("Da: " + m.getFrom()[0]));
    north.add(new JLabel("Oggetto: " +
                                        m.getSubject()));
    JPanel center = new JPanel(new BorderLayout(3, 3));
    center.add(scrollPane, BorderLayout.CENTER);
    JPanel all = new JPanel(new BorderLayout(3, 3));
    all.setBorder(new
          javax.swing.border.EmptyBorder(3, 3, 3, 3));
    all.add(north, BorderLayout.NORTH);
    all.add(center, BorderLayout.CENTER);
    getContentPane().add(all);
    // Compatto la finestra.
    pack();
    // Posiziono al centro dello schermo.
    Dimension screen = Toolkit.getDefaultToolkit(
                                    ).getScreenSize();
    Dimension frame = getSize();
    setLocation((screen.width - frame.width) / 2,
                    (screen.height - frame.height) / 2);
    // Operazione alla chiusura della finestra.
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);}
}
```

Questa classe è molto semplice. Viene assemblata una basilare GUI per mostrare il contenuto del messaggio ricevuto come argomento del costruttore. L'unico nuovo metodo che troviamo al suo interno è *getContent()* di *Message*, che naturalmente serve per recuperare il contenuto del messaggio desiderato. Mettete assieme quanto scritto sinora e compilate. L'applicazione risultante avrà aspetto e funzionalità simili a quelle mostrate in Fig. 2.

Fig. 2: *MailReceiver realizza una semplice ma funzionale GUI per la ricezione della posta elettronica.*

MAIL HTML

L'applicazione *MailReceiver* funziona sufficientemente bene con le mail di solo testo prive di allegati. Per coglierla in fallo, ad ogni modo, è sufficiente spedirsi una mail formattata con HTML. Il risultato sarà come quello mostrato in Fig. 3, con il codice HTML mostrato testualmente dal componente *MessageViewer*. Affinché l'applicazione supporti le

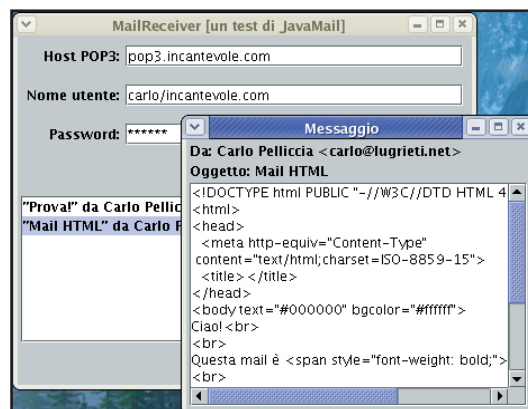



Fig. 3: *I messaggi HTML non vengono visualizzati correttamente dalla prima versione di MailReceiver.*

mail in formato HTML è necessario rivedere l'ultimo componente realizzato:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.mail.*;
class MessageViewer extends JFrame {
  // Costruttore.
  public MessageViewer(Message m) throws Exception {
    // Richiamo il costruttore della classe base.
    super("Messaggio");
    // Decodifico il messaggio.
    String text = null;
    String contentType = null;
    Object content = m.getContent();
    if (content instanceof String) {
      text = (String)content;
      contentType = m.getContentType();
      System.out.println(text);
      System.out.println(contentType);}
    // Assemblo la GUI.
    ...
    // Operazione alla chiusura della finestra.
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);}
}
```

Il metodo *getContent()* di *Message* restituisce sempre un *Object*. Il contenuto di una mail, infatti, può presentarsi in diverse forme. Nel caso tale contenuto sia una stringa, la nuova versione di *MessageViewer* non utilizzerà più una semplice *JTextArea* per la sua visualizzazione. Adesso viene sfruttato un componente di tipo *JEditorPane*, un elemento di *Swing* capace di mostrare diversi tipi di testi, tra cui HTML. Il tipo *MIME* del corpo della mail, che il componente *JEditorPane* deve necessariamente conoscere, è recuperato con il metodo *getContentType()* di *Message*.

SUPPORTO AGLI ALLEGATI

Una mail con allegati presenta il suo contenuto come un *Multipart*, vale a dire un insieme di più parti. In casi come questo, è necessario distinguere ciascuna singola parte, in modo da gestirla come meglio richiede la situazione:

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import javax.mail.*;
class MessageViewer extends JFrame {
  // Costruttore.
  public MessageViewer(Message m) throws Exception {
    // Richiamo il costruttore della classe base.
    super("Messaggio");
    // Decodifico il messaggio.
    ...
  // Salva gli allegati.
  private void save(String name, InputStream in) {
    byte[] buffer = new byte[1024];
    int l;
    FileOutputStream out = null;
    try {
      out = new FileOutputStream(name);
      while ((l = in.read(buffer, 0, buffer.length)) != -1) {
        out.write(buffer, 0, l);}
      out.close();
    } catch (IOException e) {
    } finally {
      if (out != null) try { out.close(); }
                                  catch (Exception e) {}
    }
  }
}
```

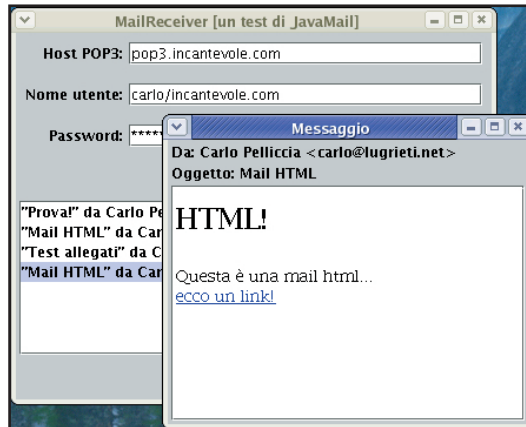



Fig. 4: Ora MailReceiver supporta i messaggi con formattazione HTML.

Quest'ultima versione di *MessageViewer* contiene un rudimentale supporto per gli allegati. Se il contenuto del messaggio risulta essere un oggetto *Multipart* anziché una semplice stringa, il nuovo codice entra in azione. Le differenti parti che compongono il contenuto vengono esaminate. Ci si aspetta che la prima di esse sia effettivamente il testo della mail. Le parti rimanenti vengono trattate come allegati e salvate su disco, nella directory da cui è stato lanciato il programma.

IL MESE PROSSIMO...

Abbiamo imparato come ricevere della posta elettronica da un server remoto, sfruttando le API JavaMail. Il mese prossimo raffineremo alcune delle tecniche qui esaminate, ed inoltre introdurremo alcuni nuovi elementi che ci saranno poi utili nella creazione del nostro client di posta "fai-da-te".

Carlo Pelliccia

Alcuni link consigliati sono:

http://www.csita.unige.it/servizi/posta/laposta.html

http://www.wowarea.com/italiano/aiuto/abmailit.htm

http://lagash.dft.unipa.it/AL/al263.htm

Potete trovare un buon tutorial di base sull'utilizzo delle API JavaMail all'indirizzo:

http://developer.java.sun.com/developer/onlineTraining/JavaMail/

Un bel po' di risposte alle domande più frequenti su JavaMail le trovate su jGuru, alla pagina:

http://www.jguru.com/faq/JavaMail

La gestione di magazzino entra nel terzo millennio

XML & C#: in pratica

In quest'ultima puntata dedicata a XML e C# vedremo come implementare riferimenti interni agli oggetti riferiti in documenti XML. Inoltre implementeremo una gestione di magazzino.

Prima di utilizzare il nostro componente di mapping, è necessario potenziarlo in modo da poter usare riferimenti interni per gli elementi *<class>*. Questa sarà la versione finale del componente che mappa codice XML in oggetti C#.

RIFERIMENTI INTERNI

Consideriamo il seguente documento:

```
<class name="Persona", id="1000">
   <field name="cognome" value="Rossi" type="string"/>
   <field name="nome" value="Mario" type="string"/>
</class>
<class name="Azienda">
   <field name="nome" value="Rossi & Bianchi"
     type="string"/>
   <classfield name="presidente" id="1000"/>
</class>
```

Quando il componente legge l'oggetto *Persona*, esso crea la persona *"Mario Rossi"* il cui *id* è *1000*. Successivamente, il componente legge l'elemento *<class name="Azienda">* e crea l'oggetto *"Rossi & Bianchi"* il cui campo presidente è un riferimento alla persona con *id* 1000 precedentemente creata, ovvero *"Mario Rossi"*. Tali riferimenti interni possono essere utili per due motivi principali:

- consentono di evitare che una stessa classe debba essere definita più volte nello stesso documento;
- consentono di definire relazioni 1 a n tra classi.

Infatti, nell'esempio precedente la classe *Persona* è stata scritta una sola volta. Inoltre se *"Mario Rossi"* è anche il presidente di un'altra azienda, sarà sufficiente aggiungere un riferimento all'*id* relativo a *"Mario Rossi"*, come mostrato di seguito:

```
<class name="Azienda">
   <field name="nome" value="Rossi Spa" type="string"/>
   <classfield name="presidente" id="1000"/>
</class>
```

Quindi, due aziende ("Rossi&Bianchi" e "Rossi Spa") hanno lo stesso presidente, cioè "Mario Rossi". Il componente, in questo caso, creerà due oggetti *Azienda* dove il campo presidente, per entrambe, è un riferimento all'unico oggetto *Persona* rappresentante *"Mario Rossi".* Passiamo quindi ad aggiungere questa nuova funzionalità al nostro componente. Il primo passo è quello di adattare lo schema XML in modo da supportare i riferimenti interni. Tale operazione è molto semplice: è sufficiente aggiungere l'attributo *id* agli elementi *<class>* e *<classfield>*, come mostrato dal seguente codice *XSD:*

```
<!—Aggiungere questa linea ai tipi Class e ClassField—>
<xs:attribute name="id" type="xs:string"/>
```

Il nuovo attributo *id* non è obbligatorio: se specificato significa che è nostra intenzione utilizzare un riferimento alla classe. Lo schema XML completo, che a questo punto è la versione finale, è presente sul CD allegato alla rivista. Il nome del file è *mapping.xsd*. Come al solito, implementeremo il *MappingFramework* e l'*Handler.* L'idea base è quella di inserire gli oggetti aventi *id* in una hash table, quindi, quando il parser leggerà un *<classField>* con *id,* l'oggetto verrà recuperato dall'hash table. Le classi *MappinFrameworkImpl* e *CompleteHandler,* versioni definitive, possono essere repertite nei listati allegati alla rivista. Esse possono essere usate per implementare un Handler concreto che legge il documento XML è mostra gli oggetti creati e relativi campi, che è proprio ciò che fa la classe *ExampleHandler*:

```
public class ExampleHandler4: ArrayHandler {
 public ExampleHandler4() :
   base(new MappingFrameworkImpl()) { }
 public override void processObject(object obj){
   Console.WriteLine(obj); } }
```

L'APPLICAZIONE D'ESEMPIO

Mostreremo come il componente per importare XML in oggetti C# possa essere integrato in una situazione reale, realizzando una semplice applica-

zione d'esempio. Essa può essere vista come un modulo che esegue un particolare *task*, in un sistema per la logistica. L'applicazione consiste in una *form* dove è mostrata la lista degli articoli presenti in magazzino e la relativa giacenza. Sistemi esterni, per mezzo di documenti XML, informeranno l'applicazione che nuovi ordini - da parte di clienti o fornitori - sono stati inoltrati. L'applicazione leggerà il documento XML, creerà gli ordini, aggiornerà la giacenza degli articoli coinvolti e registrerà il movimento di magazzino. Mediante la *form*, l'utente potrà monitorare giacenza e i movimenti (vedi Fig. 1).

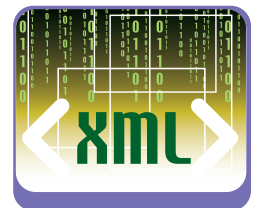
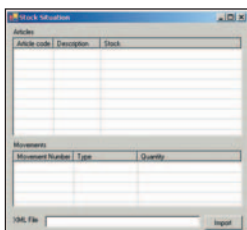
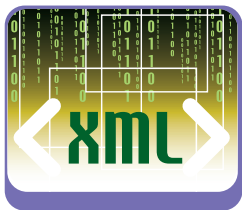
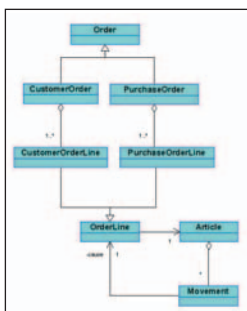



Fig. 1: ***Form dell'applicazione d'esempio.***

STRUTTURA DEL DATABASE

Il class diagram di Fig. 2 rappresenta il database utilizzato dall'applicazione d'esempio. La classe astratta *Order* (ordine) ha due generalizzazioni: *CustomerOrder* (ordine cliente) e *PurchaseOrder* (ordine d'acquisto). Il primo tipo di ordine arriva da parte di un cliente che richiede l'acquisto di determinati articoli; il secondo invece è un ordine effettuato dal magazzino per acquistare nuova merce. Entrambe le classi hanno un'aggregazione alla classe *OrderLine* (riga dell'ordine) che indica, tra le altre cose, qual'è l'articolo coinvolto e la quantità richiesta. Infatti, la classe *OrderLine* è associata alla classe *Article* (articolo), che rappresenta un articolo che può essere venduto o acquistato. Ogni elemento della classe *Article* possiede una lista di oggetti *Movement* (movimenti). L'applicazione creerà un movimento ogni volta che la giacenza dell'articolo cambia a causa di un ordine cliente o d'acquisto. Nel primo caso, la giacenza sarà decrementata poiché l'articolo viene venduto; nel secondo caso invece sarà incrementata in quanto l'articolo viene acquistato. Il codice relativo alla classe astratta *Order* è il seguente:

NOTA

LISTATI

Sul CD allegato alla rivista sono presenti sia i listati relativi a questo articolo, sia il codice delle classi implementate nei numeri precedenti e qui utilizzate.

```
public abstract class Order {
  public int number;
  public DateTime date;
  public ArrayList orderLines; }
```

Un ordine è composto da un numero (*number*), una data (*date*) e un array di righe relative all'ordine stesso (*orderLines*). Tale classe presenta due generalizzazioni: *CustomerOrder* e *PurchaseOrder*, implementate nel modo seguente:

```
public class CustomerOrder : Order {
  public string customer; }
public class PurchaseOrder : Order {
  public string supplier; }
```

Fig. 2: ***Class diagram per il database dell'applicazione d'esempio.***

Come si può notare, l'unica differenza è rappresentata dal fatto che la prima classe ha un riferimento ad un cliente (*customer*) e la seconda a un fornitore (*supplier*). Ovviamente, in un contesto reale, le differenze saranno molto più numerose. Le implementazioni delle classi *OrderLine*, *CustomerOrderLine* e *PurchaseOrderLine* sono le seguenti:

```
public abstract class OrderLine {
  public int number;
  public int quantity;
  public double price;
  public Article article;
  public abstract int getRelativeQuantity();
  public abstract string getType(); }
public class CustomerOrderLine : OrderLine{
  public override int getRelativeQuantity(){
    return -quantity; }
  public override string getType(){
    return "Customer Order Line "; } }
public class PurchaseOrderLine : OrderLine{
  public override int getRelativeQuantity(){
    return quantity; }
  public override string getType(){
    return "Purchase Order Line"; } }
```

La classe *OrderLine* ha quattro dati membro: il numero (*number*), l'articolo (*article*), la quantità in pezzi (*quantity*) e il prezzo (price). Inoltre, la classe ha anche due metodi astratti che sono implementati dalle generalizzazioni. Infatti, la classe *CustomerOrderLine* implementa *getType* in modo da restituire il tipo di riga ordine, cioè la stringa *"Customer Order Line"*. Il metodo *getRelativeQuantity* restituisce *-quantity*, poiché per un ordine cliente la merce esce dal magazzino. Allo stesso modo, per la classe *PurchaseOrderLine*, il metodo *getType* restituisce *"Purchase Order Line"* e il metodo *getRelativeQuantity* restituisce *+quantity* poiché, in questo caso, la merce è in entrata al magazzino.
La classe *Article* è la seguente:

```
public class Article {
  public string code;
  public string  description;
  public int stock;
  public ArrayList movements = new ArrayList(); }
```

Un articolo presenta un codice (*code*) e una descrizione (*description*); la variabile *stock* rappresenta la giacenza corrente nel magazzino. L'array *movements* contiene una lista di oggetti *Movement*. La classe *Movement* è realizzata come segue:

```
public class Movement {
  public int number;
  public OrderLine cause; }
```

Un movimento di magazzino ha un numero (*num-*

ber) e una causa (*cause*) che rappresenta un riferimento all'oggetto *OrderLine* che ha causato il movimento. La classe che memorizza, ricerca e gestisce gli oggetti rappresentanti la logica applicativa è chiamata *DataController*, e contiene i seguenti metodi:

```
// Aggiunge un articolo
public void addArticle(Article article)
// Trova un articolo
public Article lookupArticle(string code)
// Restituisce gli articoli
public ArrayList getAllArticles()
// Trova un ordine
public Order lookupOrder(int number)
// Aggiunge un ordine
public void addOrder(Order order)
// Importa oggetti da un file XML
// (usa XmlImporter e CompleteHandler)
public void import(string fileName)
```

Sul CD è presente l'implementazione completa della classe *DataController*.

PROGETTAZIONE E SVILUPPO DELLA GUI

Per ottenere una *form* simile a quella di Fig. 1, possiamo seguire i seguenti passi: aggiungere una nuova Window Form al progetto (all'interno del file *MainForm.cs*); impostare le seguenti proprietà della form: *Name=MainFrom, Text="Stock Situation"*; aggiungere le Label *"Articles"* e *"Movements"* al MainForm; aggiungere una *ListView* (chiamata *listArticle*) e posizionarla sotto la label "Articles"; aggiungere una *ListView* (chiamata *listMovement*) e posizionarla sotto la label *"Movements"*; aggiungere la Label *"XML File"* al MainForm; aggiungere una Text Box vicino alla Label *"XML file"* ed impostare la proprietà name a *txtXmlFileName;* aggiungere un nuovo bottone vicino la Text Box *txtXmlFileName* avente le seguenti proprietà: *Name= btnImport, Text=" Import".*
La ListView *listArticle* mostrerà tutti gli articoli presenti nel magazzino e relative giacenze. Quando l'utente clicca su una riga, nella lista movimenti (*listMovement*) saranno visualizzati tutti i movimenti di magazzino relativi all'articolo selezionato. Per rendere indipendenti la gestione delle liste e la *MainForm*, sono state implementate due classi: *ArticleListView* e *MovementListView*. Entrambe le implementazioni sono molto simili ed il codice completo può essere prelevato dai listati presenti sul CD. Di seguito riportiamo solo i dati membro e la *signature* dei metodi della classe *ArticleListView*:

```
public class ArticleListView {
  // ListView
  private ListView listView;
  // DataController
  private DataController controller;
  // Costruttore
  public ArticleListView(ListView listView,
   DataController controller){ // ... }
  public void addArticle(Article article) { // ... }
  public void reload() { // ... }
```

La classe ha i dati membro *listView*, che è un riferimento alla List View relativa agli articoli, e *controller* che è un'istanza di *DataController.* Il costruttore avrà in input entrambi *listView* e *controller.* Il costruttore, inoltre, imposta alcune proprietà di *listView* riguardanti l'intestazione delle colonne e successivamente popola la lista. La classe ha due metodi pubblici. Il primo, *addArticle*, aggiunge un articolo alla lista. Il secondo, *reload*, ripopola la lista con gli articoli forniti dall'oggetto *controller.* La *MainForm* accederà alla List View degli articoli (*listArticle*) esclusivamente mediante un'istanza di *ArticleListView (articleListView)* ed alla *List View* dei movimenti *(listMovement)* attraverso un'istanza di *MovimentListView (movementListView).* L'implementazione completa di *MovimentListView* è presente nel CD. La classe *MainForm* presenta i seguenti dati membro:

```
// DataController
private DataController controller = new DataController();
// ListView per gli articoli
private ArticleListView articleListView;
// ListView per i movimenti
private MovementListView movementListView;
```

Il metodo *MainForm_Load* creerà l'oggetto *articleListView*, come mostrato di seguito:

```
private void MainForm_Load(object sender,
  System.EventArgs e) {
  articleListView =
              new ArticleListView(listArticle, controller); }
```

Quando l'utente selezionerà un articolo, il metodo *listArticle_SelectedIndexChanged* verrà invocato. Esso creerà una nuova *MovementListView* popolata con i movimenti aggregati all'articolo selezionato:

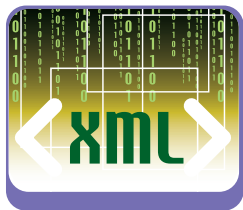
```
private void listArticle_SelectedIndexChanged
  (object sender, System.EventArgs e)
{
  if (listArticle.SelectedItems.Count != 1) return;
  string code =
    listArticle.SelectedItems[0].SubItems[0].Text;
  Article article =
    controller.lookupArticle(code);
  movementListView = new MovementListView(
                          listMovement, controller, article);
}
```

- **MAPPING XML TO C# OBJECTS USING REFLECTION**
  C#Today
  G. Naccarato
  (Wrox Press)
  Ottobre 2002

- **PROFESSIONAL C# 2ND EDITION**
  G. Naccarato
  (Wrox Press)
  2002

- **.NET SERIALIZATION**
  C#Today
  S. Gopikrishna, K. Sanghavi
  Luglio 2001
  http://www.csharptoday.com/content.asp?id=1532

- **REFLECTION IN .NET**
  C# Corner
  Hari Shankar
  Febbraio 2001
  http://www.c-sharpcorner.com/1/Reflection_in_net.asp

- **C# PROGRAMMING: ATTRIBUTES AND REFLECTION**
  XML.com,
  Jesse Liberty
  Agosto 2001
  http://www.xml.com/pub/r/1207

IMPORTARE DOCUMENTI XML

Attraverso la Text Box *txtXmlFileName* è possibile specificare il file XML da importare. Cliccando sul pulsante *"Import"* partirà il processo di mapping. Di seguito è riportato il codice relativo all'evento click del pulsante:

```
private void btnImport_Click(object sender,
 System.EventArgs e) {
 try {
  controller.import(txtXmlFileName.Text);
  articleListView.reload(); }
 catch(Exception ex){
  MessageBox.Show(ex.Message); } }
```

Come si può notare, viene invocato il metodo import della classe *DataController* e successivamente ricaricata la lista referenziata da *articleListView.* Se nessuna eccezione si verifica, sarà possibile vedere la lista degli articoli importati con i relativi movimenti. Il metodo *import* della classe *DataController* è implementato come segue:

```
public void import(string fileName) {
 XmlImporter xmlImporter =
  new XmlImporter(new ImportBO(this));
 xmlImporter.import(fileName); }
```

Esso crea un'istanza di *XmlImport,* passando un nuovo oggetto *ImportBO* e richiama il metodo *import* dell'oggetto *xmlImporter. ImportBO* è un'implementazione concreta della classe astratta *CompleteHandler:*

```
class ImportBO : CompleteHandler {
 private DataController controller;
 public ImportBO(DataController controller) :
  base(new MappingFrameworkImpl()){
  this.controller = controller;
 } // ...continua
```

Essa contiene un'istanza di *DataController (controller).* Il costruttore prende in input il *DataController* e richiama il costruttore base passando un'istanza di *MappingFrameworkImpl.* Il metodo *processObject* sarà implementato nel modo seguente:

```
public override void processObject(object obj){
 if (obj is Article) {
  importArticle((Article)obj);   }
 if (obj is Order) {
  importOrder((Order) obj); } }
```

Gli oggetti importati da questa applicazione saranno solo articoli e ordini. Il metodo delega tali operazioni ai metodi privati *importArticle* e *importOrder.* Il metodo *importArticle* aggiunge un nuovo articolo al database gestito dal *DataContoller.* Il metodo è abbastanza intelligente, infatti l'articolo è effettivamente aggiunto solo se non è già presente nel database. Il codice è il seguente:

```
private void importArticle(Article article) {
 Article stockedArticle =
  controller.lookupArticle(article.code);
 if (stockedArticle==null)
  controller.addArticle(article); }
```

Il metodo *importOrder* è un po' più complesso. In pratica, importare un ordine significa: controllare se l'ordine è già registrato; aggiungere l'ordine al database; creare un nuovo movimento per ogni riga dell'ordine; eventualmente, aggiungere al database nuovi articoli coinvolti nelle righe; aggiornare la giacenza. Il codice che effettua tali operazioni è il seguente:

```
private void importOrder(Order order) {
 Order bookedOrder =
  controller.lookupOrder(order.number);
 // Continua solo se l'ordine non esiste
 if (bookedOrder==null) {
  // Aggiunge l'ordine
  controller.addOrder(order);
  // Processa le righe dell'ordine per
  // creare i movimenti
  for (int i=0; i<order.orderLines.Count; i++) {
   OrderLine line = (OrderLine) order.orderLines[i];
   // Importa l'articolo coinvolto nella riga
   importArticle(line.article);
   line.article =
    controller.lookupArticle(line.article.code);
   // Aggiunge il movimento
   Movement movement = new Movement();
   movement.number = line.article.movements.Count + 1;
   movement.cause = line;
   line.article.movements.Add(movement);
   // Aggiorna la giacenza
   line.article.stock +=
    movement.cause.getRelativeQuantity(); }
 } }
```

L'applicazione è quindi completa. Possiamo immediatamente verificare che essa riesca realmente ad importare oggetti a partire dai documenti XML, utilizzando il documento XML *firstload.xml*, presente tra i listati di quest'articolo. Dopo averlo importato, la *form* dovrebbe avere un aspetto simile a quella di Fig. 3. La figura mostra inoltre i movimenti relativi all'articolo di codice 01005. Se volete provare il codice realizzato in questa serie di articoli, potete prelevare i listati dal CD allegato alla rivista e seguire le istruzioni presenti nel file *readme.txt.*

Giuseppe Naccarato

NOTA

RICAPITOLANDO

In questa serie di quattro articoli dedicati a XML e C# abbiamo mostrato come realizzare un componente in grado di mappare documenti XML in oggetti C#. Abbiamo usato, a tale scopo, l'oggettto XmlReader, schemi XML e reflection. Come si è visto, la versione finale del componente è stata ottenuta in quattro passi: mappando oggetti semplici, quelli complessi, array di oggetti e supportando riferimenti interni. Alla fine è stato presentato un esempio che mostra come integrare il componente all'interno delle applicazioni. Esso usa lo usa per importare articoli ed ordini da un sistema esterno verso un'applicazione .NET per la gestione della logistica.

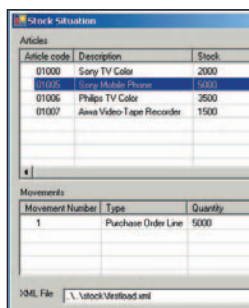



Fig. 3: La Form dopo aver importato firstload.xml.

Com'è andata la disputa dell'anno 2003

Torneo di Crobots 2003

« Parlando così, sguainò la spada affilata, / che dietro al fianco pendeva, grande e pesante, / e si raccolse e scattò all'assalto, com'aquila alto volo [...]» *(Omero, Iliade, XXII, 306-308).*

Ben ritrovati al consueto appuntamento annuale con Crobots, la competizione basata sulla programmazione di robot in linguaggio C. Omero si sarebbe trovato a suo agio, nel descrivere le gesta dei nostri prodi guerrieri fatti di bit! Crobots è oramai divenuto un evento imperdibile da diversi anni, ma forse quest'ultimo anno si è respirata un'euforia diversa, portata sicuramente dalle numerose matricole accorse alla manifestazione e, senza dubbio, motivata dalla presenza di appetitosi premi. Premi non solo messi in palio per il vincitore, ma anche esatratti a sorteggio: una formula azzeccata che rende la gara una ghiotta occasione anche per i meno "esperti" del settore. La IoMega e la Symantec hanno generosamente sponsorizzato l'evento, con succulenti premi che, resi noti sul sito e pubblicati via mailing list, che hanno attirato un'orda famelica di vecchi e nuovi crobotters. Ben 64 crobots hanno preso posto nell'arena di combattimento virtuale, per contendersi il trono di campione 2003/2004. Sfiorato il record di sempre (68 robots), si è stabilito il primato da quando l'evento è passato sotto l'egida di ioProgrammo. Qui di seguito riportiamo la lista completa dei partecipanti (Tab. 1). Innanzi tutto, dobbiamo salutare il più giovane autore di tutti i tempi: il dodicenne Martino Candon. Martino, arruolato dal babbo Roberto, si è presentato con una creatura virulenta che ha sbaragliato, nel *face2face* Micro, tutti gli avversari. Non mancano, anche quest'anno, le note di colore, come ad esempio la scelta dei nomi dei robots: Harlock con la sua Alcadia, per i trentenni nostalgici; un'impressionante ondata di Virus, pienamente in tema con il periodo autunnale; la solita orda di cavalieri Jedi per il fanatico di Guerre Stellari; il simpatico Morituro che ben sa quale fine lo attende; un'inquietante squadra inviata probabilmente dal Ministero delle Finanze, con il modulo 730, UNICO, l'ICI e via dicendo. Non c'è che dire, tutto questo rende il mondo di Crobots un luogo dove la fantasia di ciascun autore, legata alla sfrenata passione per la programmazione, ha libero sfogo.

ORGANIZZAZIONE

Il torneo è stato suddiviso in tre categorie: crobots Micro, con il limite di 500 istruzioni virtuali; crobots Midi (o Classic), con il limite di 1000 istruzioni (imposto dall'originale versione del compilatore Crobots); crobots Macro (o Big), con il limite di 2000 istruzioni virtuali. Questa

Nome Robot	Categoria	Autore	Città
730	Micro	Roberto Bevilacqua	Sanremo
Adrian	Midi	Angelo Ciufo	Latina
Aladino	Macro	Michelangelo Messina	Ischia
Alcadia	Macro	Simone Ascheri	Imperia
Ares	Midi	Alessandro De Leonardi	Imperia
Barbarian	Micro	Luca Giacometti	Cuneo
Blitz	Micro	Michelangelo Messina	Ischia
Briscolo	Micro	Alessandro De Leonardi	Imperia
Bruce	Micro	Angelo Ciufo	Latina
Cadderly	Midi	Daniele Nuzzo	Roma
Cariddi	Midi	Marco Pizzocaro	Brescia
Crossover	Macro	Michelangelo Messina	Ischia
Cyborg_2	Macro	Alessandro Carlin	Belluno
Cvirus	Micro	Martino Candon	Milano
Cvirus2	Midi	Martino Candon	Milano
Danica	Micro	Daniele Nuzzo	Roma
Dave	Macro	Angelo Ciufo	Latina
Druzil	Macro	Daniele Nuzzo	Roma
Dynacond	Midi	Lorenzo Ancarani	Ancona
Elminster	Macro	Daniele Nuzzo	Roma
Falco	Micro	Alessandro Carlin	Belluno
Foursquare	Micro	Salvo Bellerino	Catania
Frame	Micro	Salvo Bellerino	Catania
Harlock	Macro	Simone Ascheri	Imperia
Herpes	Micro	Luca Stringher	Treviso
Ici	Micro	Roberto Bevilacqua	Sanremo
Instict	Midi	Salvo Bellerino	Catania
Irpef	Midi	Roberto Bevilacqua	Sanremo
Janick	Macro	Angelo Ciufo	Latina
Janu	Micro	Maurizio Camangi	Ancona
Jedi6	Macro	Maurizio Camangi	Ancona
Knt	Macro	Marco Pizzocaro	Brescia
Kyash_3c	Midi	Franco Cartieri	Novara
Kyash_3m	Micro	Franco Cartieri	Novara
Lbr	Micro	Leonardo Baldini	Verona
Lbr1	Micro	Leonardo Baldini	Verona
Lebbra	Micro	Luca Stringher	Treviso
Maxicond	Macro	Lorenzo Ancarani	Ancona
Mg_two	Midi	Marco Garbujo	Treviso
Minicond	Micro	Lorenzo Ancarani	Ancona
Morituro	Micro	Michele Cardinale	Palermo
Nautilus	Midi	Simone Ascheri	Imperia
Nemo	Micro	Simone Ascheri	Imperia
Neo_sel	Micro	Lorenzo Ancarani	Ancona
Orione	Macro	Ale De Leonardi	Imperia
Piiico	Micro	Luca Giacometti	Cuneo
Pippo3	Micro	Andrea Creola	Novara
Pippo3b	Micro	Andrea Creola	Novara
Red_wolf	Micro	Marco Pizzocaro	Brescia
Rudolf_8	Macro	Alessandro Carlin	Belluno
Scanner	Midi	Michele Cardinale	Palermo
Scilla	Micro	Marco Pizzocaro	Brescia
Sirio	Micro	Ale De Leonardi	Imperia
Sith	Midi	Maurizio Camangi	Ancona
Sky	Midi	Alessandro Carlin	Belluno
Spaceman	Midi	Michelangelo Messina	Ischia
Tartaruga	Micro	Giuseppe Cozzolino	Napoli
Unico	Macro	Roberto Bevilacqua	Sanremo
Yoda	Micro	Maurizio Camangi	Ancona
Valevan	Micro	Valentino Bardelotto	Padova
Virus	Micro	Roberto Candon	Milano
Virus2	Midi	Roberto Candon	Milano
Virus3	Macro	Roberto Candon	Milano
Virus4	Macro	Roberto Candon	Milano

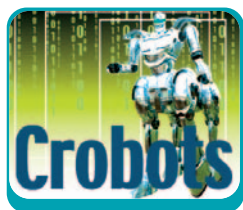
TABELLA 1: Lista robot e autori.

Pos.	Nome del Crobot	Eff. 4vs4	Eff. f2f	Eff. Totale
1	pippo3	47.71	78.13	50.76
2	nemo	47.40	76.50	50.31
3	danica	42.70	75.16	45.94
4	sirio	37.03	79.95	41.32
5	kyash_3m	33.81	71.59	37.59
6	virus	32.11	81.25	37.03
7	falco	34.22	56.58	36.45
8	janu	32.25	67.05	35.73
9	pippo3b	29.62	43.41	31.00
10	minicond	28.76	45.07	30.39
11	730	25.57	69.95	30.01
12	yoda	26.74	58.78	29.94
13	red_wolf	27.23	51.63	29.67
14	scilla	27.80	37.07	28.73
15	ici	24.33	54.98	27.40
16	cvirus	19.89	89.15	26.82
17	blitz	22.51	52.08	25.46
18	neo_sel	22.10	47.46	24.63
19	bruce	20.05	52.11	23.25
20	tartaruga	13.70	49.79	17.31
21	herpes	14.46	21.04	15.12
22	piiico	10.71	54.47	15.08
23	briscolo	13.95	22.82	14.84
24	lebbra	13.41	18.59	13.93
25	morituro	11.92	14.02	12.13
26	valevan	9.99	19.75	10.97
27	barbarian	3.56	60.35	9.24
28	lbr	7.43	13.57	8.04
29	lbr1	5.22	27.65	7.46
30	frame	4.78	8.41	5.14
31	foursquare	1.81	1.32	1.76

TABELLA 2: Finale categoria Micro

formula, già sperimentata lo scorso anno, si è rivelata estremamente fortunata. Le tre diverse categorie permettono allo sviluppatore di cimentarsi con robots di "peso" consono alle proprie capacità e alla propria esperienza di programmazione. Ciascuna categoria, a partire dalla Midi, ospita i robot di quella inferiore, aumentando di volta in volta il numero dei combattenti. Le pagine dei risultati sono state pubblicate sul sito ufficiale di Crobots, ospitato dal portale di ioProgrammo, all'indirizzo *www.ioprogrammo.net/crobots*.
Un doveroso ringraziamento va alla Web Agency *Adria Lab* (*www. adrialab.it*), che ha messo cortesemente a disposizione due server, basati su Debian Linux, gestiti per l'occasione da Maurizio Camangi: il primo, un Celeron 600 (soprannominato muletto), in diurna; il secondo, un potente bi-processore Pentium, in notturna. Entrambi, per un totale di circa 6,3 milioni di SGNIPs (buffa sigla per identificare il numero di istruzioni virtuali Crobots al secondo), hanno calcolato incessantemente tutti gli scontri per diverse ore, generando alcuni gigabyte di logfiles. Una serie di script shell creati appositamente allo scopo, insieme alle indispensabili utility Count e Torneo XP, di Simone Ascheri e Michelangelo Messina, hanno aggiornato automaticamente l'andamento del torneo, notificando i risultati via e-mail agli iscritti delle mailing list di Crobots su ioProgrammo e Yahoo!, e aggiornando via ftp il sito ufficiale di Crobots, con aggiornamenti periodici effettuati ad ogni avanzamento del torneo pari al 5%.

SVOLGIMENTO E RISULTATI FINALI

Per il torneo Micro è stato sufficiente un unico girone, composto da 31 mostriciattoli, per decretare il vincitore di categoria Pippo3 di Andrea Creola. Senza alcun dubbio questa è stata una meritata vittoria: è dunque d'obbligo un tributo a un veterano della manifestazione che, con la famosa serie di robots Pippo ci segue instancabilmente dal lontano 1992. Il torneo Midi, con all'avvio 47 creature, ha necessitato di due gironi di qualificazione e di un girone di ripescaggio, per eleggere i 32 finalisti. La finale Midi ha poi visto trionfare... un robot Micro! Simone Ascheri, con il suo Nemo, ha posto un importante sigillo, con questa spettacolare vittoria, nella storia della programmazione Classic (fino a 1000 istruzioni). Per il torneo Macro è stato utilizzato un doppio turno con gironi di qualificazione, semifinali e ben due gironi di ripescaggio: una durissima selezione che, da 64 robots, ha decretato i 32 finalisti.

Pos.	Nome del Crobot	Eff. 4vs4	Eff. f2f	Eff. Totale
1	nemo	41.07	69.31	45.78
2	cadderly	40.79	67.30	45.21
3	nautilus	40.48	65.51	44.65
4	pippo3	37.26	60.88	41.20
5	danica	37.65	56.92	40.86
6	cariddi	36.47	60.31	40.45
7	virus2	29.27	68.96	35.88
8	spaceman	30.17	60.83	35.28
9	irpef	28.18	56.17	32.84
10	sirio	25.00	64.54	31.59
11	sky	24.79	58.56	30.42
12	virus	22.92	63.24	29.64
13	sith	23.95	57.99	29.62
14	pippo3b	26.33	30.78	27.07
15	kyash_3c	21.45	54.90	27.03
16	ares	22.49	48.02	26.75
17	falco	23.80	32.16	25.20
18	kyash_3m	20.20	49.96	25.16
19	scilla	25.07	22.86	24.70
20	adrian	19.61	41.62	23.28
21	730	18.29	47.62	23.18
22	dynacond	21.32	31.48	23.01
23	minicond	21.23	31.85	23.00
24	cvirus2	11.37	76.96	22.31
25	cvirus	10.64	74.88	21.34
26	red_wolf	19.70	28.23	21.13
27	janu	15.41	41.15	19.70
28	ici	15.93	31.40	18.51
29	yoda	15.32	33.92	18.42
30	neo_sel	14.88	27.33	16.96
31	blitz	13.82	23.90	15.50
32	bruce	11.46	27.15	14.07

TABELLA 3: Finale categoria Midi.

A questo punto non possiamo che fare i nostri complimenti, a nome degli organizzatori e di tutta la comunità "crobotica", al vincitore assoluto della competizione: Simone Ascheri con il suo Alcadia! Tre robots, di cui uno Micro e già campione nella categoria Midi, sul podio! Ad occhio e croce, mai si era vista una prestazione del genere. Gli altri autori si uniscono nel coro dei complimenti, senza far mancare qualche simpatica battuta sul codice "autocriptante" scritto da Simone, comprensibile soltanto dal compilatore Crobots.

Pos.	Nome del Crobot	Eff. 4vs4	Eff. f2f	Eff. Totale
1	alcadia	41.53	62.95	45.10
2	harlock	39.51	62.81	43.40
3	nemo	38.64	62.85	42.67
4	elminster	38.33	62.99	42.44
5	druzil	37.78	60.37	41.54
6	nautilus	37.73	56.06	40.78
7	cadderly	33.60	61.31	38.21
8	danica	33.63	44.73	35.48
9	aladino	27.74	65.32	34.00
10	crossover	26.64	64.30	32.92
11	virus3	26.91	57.46	32.00
12	virus4	23.82	57.68	29.47
13	spaceman	24.27	52.07	28.90
14	sirio	22.90	58.93	28.90
15	virus2	22.83	57.80	28.66
16	pippo3	22.23	51.86	27.16
17	rudolf_8	20.53	52.77	25.90
18	cariddi	19.37	46.90	23.96
19	irpef	18.81	46.55	23.44
20	sky	18.60	46.39	23.23
21	knt	18.16	46.85	22.95
22	orione	20.02	34.80	22.48
23	ares	19.27	35.11	21.91
24	virus	15.82	52.27	21.90
25	cyborg_2	15.70	48.89	21.23
26	janick	18.45	31.02	20.54
27	jedi6	15.09	46.40	20.31
28	sith	13.38	46.28	18.87
29	730	12.29	34.08	15.92
30	falco	14.03	19.06	14.86
31	scilla	13.45	15.47	13.79
32	red_wolf	10.40	16.78	11.47

TABELLA 4: Finale assoluta categoria Macro.

CONCLUSIONI

Per chiudere in bellezza non possiamo che lasciare la parola al nostro illustre nonché modestissimo neo-campione Simone Ascheri:

«Devo dire che sono rimasto decisamente sorpreso dall'esito del torneo 2003. Se, fino ad un mese dalla competizione, nutrivo qualche speranza in tutte e tre le categorie, dopo gli ultimi risultati raggiunti da Daniele avevo decisamente ridimensionato le mie aspettative.
Non potevo migliorare ancora i miei robot: un triangolo è un triangolo, lo puoi allungare e schiacciare, ma tante altre variazioni sul tema non si possono proporre. Ho quindi cercato una strada di sviluppo alternativa, incentrata su due considerazioni: mi era difficile scrivere una routine finale generale che funzionasse bene contro ogni avversario; se tutti i miei robot fossero arrivati alla finale, mi sarebbe stato noto il comportamento di 1/8 degli avversari. Ho così preparato quattro varianti dello stesso robot, differenziate dall'attacco finale, e trascorso gli ultimi 15 giorni a farle 'collaborare'. Viste le premesse, sono rimasto stupefatto prima dalla vittoria di Nemo, microbo, tra i classic e, ancora di più, dall'affermazione di Alcadia nella competizione generale.
Credo di essere stato favorito dalla decisione di Daniele di utilizzare l'attacco triangolare, vera novità di questa edizione, solo nel medio gioco, dal massiccio ritorno all'uso delle Toxiche e dagli avversari incontrati: in un torneo combattuto come questo anche la differenza di uno a due robot avrebbe potuto cambiare il volto della finale. Doverosi ringraziamenti vanno a: Daniele Nuzzo e al suo Drizzt, Andrea Creola (meritatamente microcampione) per Pippo2a, Alessandro Carlin (per non avere avuto il tempo di sviluppare seriamente), Michelangelo Messina per avermi fatto capire che l'ottimizzazione esasperata non è cosa buona.»

Includiamo qui di seguito un estratto tecnicamente molto interessante, relativo ad una delle funzioni di fuoco, direttamente dal sorgente di Alcadia:

```
/*Gli 0 inseriti nel codice servono per migliorare la
                         temporizzazione dello sparo.*/
Fuoco(dove) {
   drive (dove,100);
   if (oldr=scan(a,10))  {
        if (scan(a,2)){if ((cannon(a+0+0+0+0+0,
                         3*scan(a,10)-2*oldr))return;}
        else if (scan(a-=7,5)){if ((cannon(a-6+0+0,
                         2*scan(a,10)-oldr))) return;}
        else if (scan(a+=14,5)){
          if((cannon(a+7,2*scan(a,10)-oldr))) return;}
        else if (scan(a+=10,5)){
          if((cannon(a+7,2*scan(a,10)-oldr))) return;}
          else a+=15;}
      else if (rng=scan(a+=340,10)) return (cannon(a,rng));
      else if (rng=scan(a+=40,10))  return
                                       (cannon(a,rng));
      else if (rng=scan(a+=300,10)) return
                                       (cannon(a,rng));
      else if (rng=scan(a+=80,10))  return
                                       (cannon(a,rng));
      else if (rng=scan(a+=260,10)) return
                                       (cannon(a,rng));
      else if (rng=scan(a+=120,10)) return
                                       (cannon(a,rng));
      else if (rng=scan(a+=220,10)) return
                                       (cannon(a,rng));
      else if (rng=scan(a+=160,10)) return
                                       (cannon(a,rng));
      else if (rng=scan(a+=180,10)) return
                                       (cannon(a,rng));
          else a+=270;
          return Fuoco(dove);
}
```

Per tutto il resto vi rimandiamo sul sito ufficiale *http://www.ioprogrammo.it/crobots* al link Tournament per i risultati dettagliati, e al link Download per scaricare i sorgenti dei robots.
Vi aspettiamo ovviamente in mailing list per assistere all'epico scontro fra tutti i robots scritti dal 1991 fino ad oggi!

Maurizio Camangi
Simone Ascheri
Michelangelo Messina

I trucchi del mestiere

Tips&Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

ACQUISIRE IMMAGINI TRAMITE SCANNER

Il tip mostra come dotare le proprie applicazioni di funzionalità per acquisire documenti da una fonte tipo scanner.

Tip fornito dal sig. D.Spinoglio

Ecco i primi passi da seguire:

1) Creare il form.
2) Aggiungete i componenti *"Controllo digitalizzazione immagine Kodak"* e *"Controllo modifica immagine Kodak"*
3) Aggiungere al form i componenti selezionati nel passaggio precedente (*ImgScan* e *ImgEdit*), inoltre: un bottone ed una label nel form
4) Copiate e incollate il seguente codice:

```
Private Sub Form_Load()
Command1.Caption = "Scannerizza!"
Label1.Caption = "Staus: none"
End Sub

Private Sub Command1_Click()
' scanner pronto?
ImgScan1.ScannerAvailable
' apre la porta dello scanner
ImgScan1.OpenScanner
' inizia la scansione
ImgScan1.StartScan
End Sub

Private Sub ImgScan1_PageDone(ByVal PageNumber As Long)
Label1.Caption = "status: page " & PageNumber & " done."
'salva l'immagine al percorso specificato
ImgEdit1.SaveAs "c:\provascan.jpg", 6
End Sub

Private Sub ImgScan1_ScanDone()
Label1.Caption = "status: scan done."
End Sub

Private Sub ImgScan1_ScanStarted()
Label1.Caption = "status: scan started."
End Sub
```

5) Ora eseguite la vostra applicazione e godetevi lo spettacolo.

Di seguito sono proposte alcune proprietà utili del componente *ImgEdit:*

ImgEdit1.ZoomToSelection - ingrandisce la selezione.
ImgEdit1.Flip - capovolge l'immagine.
ImgEdit1.ClipboardCopy - copia il contenuto della selezione nella Clipboard.
ImgEdit1.ClipboardCut - taglia e copia il contenuto della selezione nella Clipboard.
ImgEdit1.FitTo - ridimensiona l'immagine a seconda della grandezza del componente. Le opzioni sono:

1) dimensione reale;
2) lo adatta al controllo
3) dimensioni doppie.

ImgEdit1.PrintImage - stampa l'immagine.
ImgEdit1.RotateLeft - ruota l'immagine a sinistra di 90°.
ImgEdit1.RotateRight - ruota l'immagine a destra di 90°.

UN SEMPLICE AUTOCOMPLETE DEI FORM DI IE

Il tip cerca tra i processi attivi le istanze di IE, e se ne trova, inizia a ricercarvi, all'interno, gli eventuali controlli di tipo text e password. A questo punto li riempie con dei valori decisi a priori. Lo scopo di questo tip è mostrare come sia possibile implementare un semplice autocomplete dei form per Internet Eplorer.

Tip fornito dal sig. L.Cecchetto

```
Option Explicit

Private winTitolo As String

'Verifico se l'oggetto passatomi e' un campo di tipo password
Private Function IsPasswordBox(Elemento As Object) As Boolean
    On Error GoTo err_password
    If LCase(Elemento.getAttribute("Type")) = "password" Then
        IsPasswordBox = True
```

```
    Else
        IsPasswordBox = False
    End If
    Exit Function
err_password:
    IsPasswordBox = False
End Function

'Verifico se il campo e' una text box
Private Function IsTextBox(Elemento As Object) As Boolean
    On Error GoTo err_text
    If LCase(Elemento.getAttribute("Type")) = "text" Then
        IsTextBox = True
    Else
        IsTextBox = False
    End If
    Exit Function
err_text:
    IsTextBox = False
End Function
Private Function CercaCampi(Documento As Object) As Boolean
    Dim Elemento As Object
    Dim numOggetti As Long
    Dim indiceOggetti As Long
    Dim Trovato As Boolean
    Dim ok As Integer
    'Prendo il numero degli oggetti nel documento
    numOggetti = Documento.All.length
    'Scorro gli elementi fino a trovarne uno di tipo password o text
    For indiceOggetti = 0 To numOggetti - 1
        DoEvents
        Set Elemento = Documento.All.Item(indiceOggetti)
        'Verifico se e' una password-box e la riempio con la parola pluto
        If IsPasswordBox(Elemento) Then
          'Il false serve per rendere case-insensitive la ricerca
                                                    dell'attributo value
            ok = Elemento.setAttribute("Value", "pluto", False)
            Trovato = True
        End If
        'Verifico se e' una text-box e la riempio con la parola paperino
        If IsTextBox(Elemento) Then
            'Il false serve per rendere case-insensitive la ricerca
                                                    dell'attributo value
            ok = Elemento.setAttribute("Value", "paperino", False)
            Trovato = True
        End If
    Next
    numOggetti = Documento.frames.length
```

IL TIP DEL MESE

UNA CONSOLE PER LE VOSTRE APPLICAZIONI

Vi è mai capitato di aver bisogno di un console per il programma che state scrivendo? Magari soltanto per scrivere delle righe di log…ecco il tip che consente di implementarla…

Tip fornito dal sig. M.Catena

```
Private Declare Function AllocConsole Lib "kernel32" () As Long
Private Declare Function FreeConsole Lib "kernel32" () As Long
Private Declare Function CloseHandle Lib "kernel32" (ByVal
                                    handleObj As Long) As Long
Private Declare Function GetStdHandle Lib "kernel32" (ByVal
                                       handle As Long) As Long
Private Declare Function WriteConsole Lib "kernel32" Alias "WriteConsoleA"
        (ByVal hConsoleOutput As Long, lpBuffer As Any, ByVal
      nNumberOfCharsToWrite As Long, lpNumberOfCharsWritten
                        As Long, lpReserved As Any) As Long

Private Const STD_OUTPUT_HANDLE = -11&

Dim hConsole As Long

Private Sub btnTest_Click()
    Dim retCode As Long, stdOut As String, byteScritti As Long
    stdOut = vbCrLf & "Grazie per aver provato !!!" & vbCrLf
    retCode = WriteConsole(hConsole, ByVal stdOut, Len(stdOut),
                                                byteScritti, ByVal 0&)
    Shell App.Path & "\TEST.BAT"
End Sub

Private Sub Form_Load()
    hConsole = IstanziaConsole
  End Sub

  Private Sub Form_Unload(Cancel As Integer)
     RilasciaConsole hConsole
  End Sub
Function IstanziaConsole() As Long
    IstanziaConsole = 0
    If AllocConsole() Then
        IstanziaConsole = GetStdHandle(STD_OUTPUT_HANDLE)
        If IstanziaConsole = 0 Then
            MsgBox "Impossibile reindirizzare l'output sulla Console",
                                               vbOKOnly + vbCritical
        End If
    Else
        MsgBox "Impossibile aprire la Console", vbOKOnly +
vbCritical
    End If

End Function
Function RilasciaConsole(handleConsole As Long)
     CloseHandle handleConsole
     FreeConsole
End Function
```

```
    'Eseguo la verifica anche su eventuali frame nella pagina
    For indiceOggetti = 0 To numOggetti - 1
        'Esegui la ricerca anche in questi frame
        If CercaCampi(Documento.frames.Item(indiceOggetti).document)
                                                    Then Trovato = True
    Next
    CercaCampi = Trovato
End Function
Private Sub Scansiona()
    Dim objShellWins As New SHDocVw.ShellWindows
    Dim objExplorer As SHDocVw.InternetExplorer
    Dim Documentoument As HTMLDocument
    Dim Trovato As Boolean
    Dim Eseguito As Boolean
    Screen.MousePointer = vbHourglass
    'Scorri tutte le fineste aperte
    For Each objExplorer In objShellWins
        If TypeOf objExplorer.document Is HTMLDocument Then
            Set Documentoument = objExplorer.document
            'Salva il titolo cosi' da poterle riconoscere
            winTitolo = Documentoument.Title
            'Comincia la ricerca nel documento
            Eseguito = CercaCampi(Documentoument)
            If Eseguito Then Trovato = True
        End If
    Next
    Screen.MousePointer = vbDefault
End Sub
Private Sub cmdPasswords_Click()
    Scansiona
End Sub
```

SALVARE UN GRAFICO MSCHART COME IMMAGINE

Il tip consente il salvataggio di un grafico MSChart come immagine, il tutto senza fare uso di API sistema.

Tip fornito dal sig. A.Piana

```
Public Sub SalvaMSChart(Graph As MSChart, pic As PictureBox,
                                        ByVal FileName As String)
'Copia il grafico nella clipboard
Graph.EditCopy
' Imposta la picture della PicBox
pic.Picture = Clipboard.GetData(vbCFMetaFile)
' Salva l'immagine
SavePicture pic.Picture, FileName
End Sub
```

JAVA

GESTIONE AVANZATA ECCEZIONI!

La gestione delle eccezioni è sempre un problema, con questo mini framework è possibile proteggere il codice "pericoloso" in un'apposita struttura, che faciliterà la gestione delle eccezioni o che permetterà di utilizzare metodi, con eccezioni *checked* senza il blocco *try/catch* e senza perdere l'eventuale eccezione scatenata!

Tip fornito dal sig. S.Fago

```
/**
* Classe d'utilità per uso dei TryMethods...
*/
public class ExceptionUtil {
  private ExceptionUtil() {}

  public static Object execute(TryMethod method,java.util.Collection
                                                              args){
   try {
          return method.execute(args);
       }
       catch (Exception ex) {
          return ex;
       }
    }
}//end

/**
 * Interfaccia che stabilisce il contratto di un TryMethod da
                                                implementare per le
 * RuntimeException...
 */
public interface TryMethod{
   public Object execute(java.util.Collection args);
}

/**
* Implementazione di TryMethod per le checked exception...
*/
public class ReflectiveMethod implements TryMethod{
    private Object target;
    private String methodName;
    private Class[] paramTypes;

    public void init(Object target, String methodName, Class []
                                                       paramTypes){
     this.target = target;
     this.methodName = methodName;
     this.paramTypes = paramTypes;
    }

    public Object execute(java.util.Collection args){
     try {
          Class clazz = target instanceof
                              Class?(Class)target:target.getClass();
          return (clazz.getDeclaredMethod(methodName,paramTypes)
        ).invoke(  (target instanceof Class?null:target),args.toArray());
     }catch (java.lang.reflect.InvocationTargetException iex){return
           iex.getTargetException();}catch (Exception ex) {return ex;}
    }
  }//end
```

UN SEMPLICE BLOCCO NOTE

La class java allegata implementa un semplice blocco note, tipo quello di Windows, capace di gestire, grazie ai flussi di I/O messi a disposizione da java, i file di testo (carica/salvataggio).Nello sviluppo dell'applicazione si è deciso di non utilizzare massicciamente le opzioni grafiche delle *JFrame*, questo per mettere in rilevanza le potenzialità dei flussi di java.

Tip fornito dal sig. A.Avolio

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
public class BloccoNotes implements WindowListener, ActionListener
{
  //Istanze della classe
  private Frame finestra;
  private TextArea areaTesto;
  //Costruttore della classe
   public BloccoNotes(Frame f, TextArea t)
   {
     finestra = f;
     areaTesto = t;
   }
  //Gestione della grafica
  // Metodi non implementati dell'interfaccia WindowListener
  public void windowActivated(WindowEvent evt) {}
  public void windowDeactivated(WindowEvent evt) {}
  public void windowDeiconified(WindowEvent evt) {}
  public void windowIconified(WindowEvent evt) {}
  public void windowOpened(WindowEvent evt) {}
  // Metodi implementati dell'interfaccia WindowListener
  public void windowClosed(WindowEvent evt)
  {
     System.exit(0);
  }
  public void windowClosing(WindowEvent evt)
  {
     finestra.dispose();
  }
  // Metodo acoltatore ha il computo di ascoltare i comandi
  public void actionPerformed(ActionEvent evt)
  {
    String comando = evt.getActionCommand();
    //gestione dei vari eventi
    if (comando.equals("Esci"))
    {
       finestra.dispose();
    } else if (comando.equals("Apri"))
    {
       areaTesto.setText("");
       FileDialog d = new FileDialog(finestra, "Apri documento",
                                                   FileDialog.LOAD);
       d.setVisible(true);
       if (d.getFile() != null)
       {
         try
         {
           String line;
           BufferedReader in = new BufferedReader(new FileReader(
                   d.getDirectory() + File.separator + d.getFile()));
           while ((line = in.readLine()) != null)
             areaTesto.append(line + "\n");
         } catch (Exception e)
         {
           System.out.println("Errore: " + e);
         }
       }
    }
    else if (comando.equals("Salva con nome..."))
    {
      FileDialog d = new FileDialog(finestra, "Salva documento con
                                              nome", FileDialog.SAVE);
     d.setVisible(true);
     PrintWriter fout;
     String Stringa=areaTesto.getText();
     try
     {
        fout=new PrintWriter(new FileWriter(d.getDirectory()
                                     +File.separator+d.getFile()));
        StringTokenizer st=new StringTokenizer(Stringa,"\n"); //E'
                     importante per gestire la pressione dell'invio
        while (st.hasMoreTokens())
          fout.println(st.nextToken());
        fout.close();
     }
     catch(Exception e)
     {
     }
    }
}
  /** Metodo Main */
  public static void main(String[] args)
  {
    Frame f = new Frame("My blocco note ");
    TextArea t = new TextArea();
    BloccoNotes n = new BloccoNotes(f, t);
    MenuBar mb = new MenuBar();
    // Preparazione menù file il menu File
    Menu menu = new Menu("File");
    mb.add(menu);
    menu.add("Apri");
    menu.add("Salva con nome...");
    menu.addSeparator();
    menu.add("Esci");
    menu.addActionListener(n);
    // Prepara la finestra
    f.addWindowListener(n);
    f.setMenuBar(mb);
    f.add(t);
    f.setSize(400, 500);
    f.setVisible(true);
  }
}
// BloccoNotes
```

PASSARE VALORI TRA PAGINE WEB

Il codice JavaScript permette di passare i valori di variabili tra pagine senza l'uso di php, cgi o cookies semplicemente sfruttando l'url della pagina. La pagina di "partenza" quando chiama la pagina “ricevente" il valore deve rispettare la seguente sintassi *"indirizzo?variabile1=valore1&variabile2=valore2&variabile3=valore3"*, ovvero quella che, generalmente, viene utilizzata dai form utilizzando il metodo "GET". Per poter ottenere il valore di una variabile è sufficiente chiamare la funzione leggivarabile passandole come parametro semplicemente il nome della variabile stessa. In allegato un file ".*htm*" d'esempio che mostra il funzionamento dello script anche in coppia con i form.

Tip fornito dal sig. I. Scarabello

```
function leggivariabile(variabile) {
        // variabile contiene il nome della variabile
    var indirizzo = window.location.toString() + "&";
        // indirizzo contiene l'indirizzo della pagina nella forma
        // "indirizzo?variabile1=valore1&variabile2=
                                valore2&variabile3=valore3&"
    var posizionenome=indirizzo.indexOf(variabile + "=");
        // trova la posizione in cui e' memorizzata la variabile
    if (posizionenome==-1) return "";
        // se restituisce -1 la variabile non c'e'! :(
            var sottostringa=indirizzo.substring(posizionenome);
        // taglia tutto quello che c'e' prima della variabile
    var inizio=sottostringa.indexOf("=");
        // trova l'uguale dopo il quale si trova il valore
    var fine=sottostringa.indexOf("&");
        // trova la & che indica dove finisce il valore
    var variabile=sottostringa.substring(inizio + 1, fine);
        // assegna a variabile il valore contenuto tra = e &!
    return variabile;
}
```

C++

UN CATTURA SCHERMO FACILE FACILE...

Il tip in allegato illustra come effettuare la cattura dello schermo in C# (utilizzando l'api di Windows *BitBlt*) e salvare l'immagine catturata in un file bitmap.

Tip fornito dal sig. A. Pelleriti

```
using System;
using System.Drawing;
using System.Runtime.InteropServices;

class CatturaSchermo
{
  [DllImport("gdi32.dll")]
 public static extern bool BitBlt(IntPtr hdcDst, int xDst, int yDst, int
          cx, int cy, IntPtr hdcSrc, int xSrc, int ySrc, uint ulRop);

 public static void Main(string[] args)
 { string filename=@"C:\cattura.bmp";
   if(args.Length>0)
     filename=args[0];
     Graphics gfxScreen = Graphics.FromHwnd(IntPtr.Zero);
     Bitmap bmp = new Bitmap((int) gfxScreen.VisibleClipBounds.Width,
            (int) gfxScreen.VisibleClipBounds.Height, gfxScreen);
    Graphics gfxBitmap = Graphics.FromImage(bmp);
    IntPtr hdcScreen = gfxScreen.GetHdc();
    IntPtr hdcBitmap = gfxBitmap.GetHdc();
    BitBlt(hdcBitmap, 0, 0, bmp.Width, bmp.Height, hdcScreen,
                   0, 0, 0x00CC0020);//costante SRCCOPY
    gfxBitmap.ReleaseHdc(hdcBitmap);
    gfxScreen.ReleaseHdc(hdcScreen);
   gfxBitmap.Dispose();
   gfxScreen.Dispose();
   bmp.Save(filename);
 }
}
```

Per provare il tip basta salvare il codice in un file (ad es. cattura.cs) e compilare con il comando

```
csc cattura.cs
```

Quindi eseguire con il comando:

```
cattura.exe [percorsofile.bmp]
```

Nel caso in cui non si fornisca il nome del file verrà creato un file *c:\cattura.bmp*.

Delphi

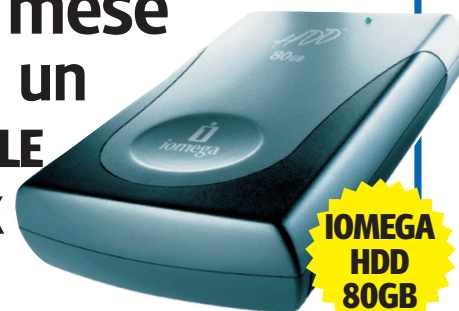
REPERIRE INFORMAZIONI UTILIZZANDO IL REGISTRO DI SISTEMA

Il TIP serve a reperire informazioni utili (quali nome utente , nome bios, sheda madre ed ecc...) dal registro di Windows.Per fare tutto ciò bisogna creare una nuova applicazione e nel Form inserire una ListBox e successivamente negli USES inserire la libreria *Registry*.

Tip fornito dal sig. V. Dentamaro

Doppio click sul form dell'applicazione e nella procedura di creazione del form inserire:

```
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
reg : tregistry;
s :string;
i :integer;
begin
  Reg := TRegistry.Create ;
    Reg.RootKey :=HKEY_CURRENT_USER;
    if reg.OpenKey('Software\Microsoft\Active Setup\Installed
      Components\{44BBA840-CC51-11CF-AAFA-00AA00B6015C}',
                                                  false )then
  begin
    s:=reg.ReadString('Username');
    l.Items.Add('Nome Utente   = '+s);
  end;
 Reg := TRegistry.Create ;
  Reg.RootKey :=HKEY_CURRENT_USER;
  //
  if reg.OpenKey('Software\Microsoft\Windows\ShellNoRoam',
                                                  false )then
  begin
    s:=reg.ReadString('');
    l.Items.Add('Nome Compuiter = '+s);
  end;
  Reg := TRegistry.Create ;
   Reg.RootKey :=HKEY_LOCAL_MACHINE;
   //
   if reg.OpenKey('HARDWARE\DESCRIPTION\System\
                                  CentralProcessor\0',false )then
     begin
       s:=reg.ReadString('ProcessorNameString');
       l.Items.Add('Processore = '+s);
     end;
    Reg := TRegistry.Create ;
     Reg.RootKey :=HKEY_LOCAL_MACHINE;
     //
     if reg.OpenKey('HARDWARE\DESCRIPTION\
                           System\CentralProcessor\0',false )then
       begin
         s:=reg.ReadString('Identifier');
         l.Items.Add('Identificazione  = '+s);
       end;
      Reg := TRegistry.Create ;
       Reg.RootKey :=HKEY_LOCAL_MACHINE;
       //
       if reg.OpenKey('HARDWARE\DESCRIPTION\
                           System\CentralProcessor\0',false )then
        begin
          i:=reg.Readinteger('~MHz');
          l.Items.Add('Velocit‡ CPU  =  '+inttostr(i)+'  MHz');
        end;
       Reg := TRegistry.Create ;
        Reg.RootKey :=HKEY_USERS;
         //
        if reg.OpenKey('S-1-5-21-1220945662-1454471165-
               725345543-1003\Software\Liter\SystemInfo\BIOS',
                                                  false )then
          begin
            s:=reg.ReadString('Product');
            l.Items.Add('Nome Bios  = '+s);
            end;
        Reg := TRegistry.Create ;
          Reg.RootKey :=HKEY_USERS;
          //
         if reg.OpenKey('S-1-5-21-1220945662-1454471165-
               725345543-1003\Software\Liter\SystemInfo\BIOS',
                                                  false )then
              begin
             s:=reg.ReadString('Date');
             l.Items.Add('Data  = '+s);
            end;
        Reg := TRegistry.Create ;
             Reg.RootKey :=HKEY_USERS;
          //
      if reg.OpenKey('S-1-5-21-1220945662-1454471165-
                     725345543-1003\Software\Liter\SystemInfo\
                                        Mainboard',false )then
         begin
           s:=reg.ReadString('Current');
           l.Items.Add('Nome scheda madre   = '+s);
         end;
         //ed eccetera................
end;
```

C++ e Delphi: applicazioni di robotica

Facciamo il caffè con il Robot

L'applicazione di quanto verrà discusso in queste pagine ci permetterà di programmare il braccio meccanico per l'esecuzione di compiti complessi.

La vita quotidiana, unitamente a milioni di anni di evoluzione, ci hanno ormai abituati a svolgere tutta una serie di azioni, anche complesse, con semplicità e naturalezza. Anche preparare un caffè può sembrare una operazione semplice, ferma restando la certezza che per farne uno buono occorre sconfinare nell'arte: proviamo ora ad immaginare come si possa 'insegnare' ad una apparecchiatura elettromeccanica (il nostro Robot) tutta quella sequenza di movimenti forti e leggeri, lenti e veloci, operati nello spazio che ci portano ad assaporare la poesia di un buon caffè. Ovviamente quello che stiamo facendo è un puro esempio, che ha lo scopo di introdurci alla programmazione tridimensionale di un Robot. L'approccio alla programmazione di un Robot è fortemente dipendente dal problema da risolvere. Gli algoritmi di manipolazione sono tanto più complessi quanto maggiore è la flessibilità che si desidera dalla macchina. Nel caso ad esempio di un Robot industriale che disponga in un contenitore gli oggetti che scorrono su un nastro trasportatore potrà essere richiesto un grado di 'intelligenza' diverso a seconda se gli venga richiesto semplicemente di scartare gli oggetti che non sono posizionati correttamente per la manipolazione, oppure se debba correggere l'errore di posizione. La prima opzione comporta soltanto la ripetizione di una serie di azioni che possono essere programmate attraverso diversi ausili. Nel secondo caso è invece necessario un sistema avanzato di riconoscimento della posizione degli oggetti ed un sofisticato algoritmo di manipolazione. Procedendo per gradi nella trattazione di questo affascinante aspetto della programmazione dei sistemi di manipolazione, vogliamo fornire al lettore uno strumento hardware realmente realizzabile, completo di schemi elettrici, nonché il relativo software compilato e direttamente utilizzabile e dotato di tutti i codici sorgenti, scritti in C++ per la gestione dei movimenti di rotazione e flessione del polso del braccio meccanico. Al temine di queste pagine saremo in grado di programmare il braccio meccanico e di porlo di fronte ad una tazzina di caffè con il compito di mescolarne il contenuto, senza rompere o rovesciare il contenitore.

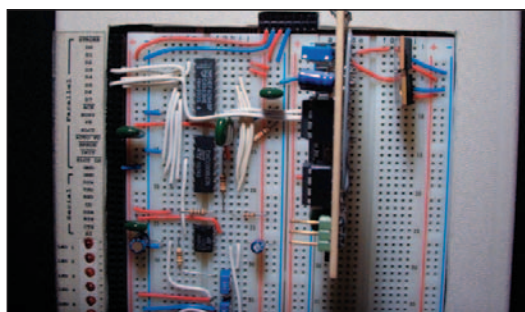
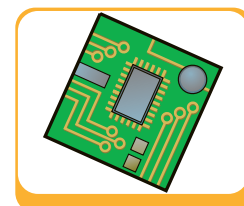



Fig. 1: *In queste pagine vedremo come comandare un braccio meccanico attraverso la registrazione e la riproduzione dei suoi movimenti.*

Fig. 2: *L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys.*

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo: luca.spuntoni@ioprogrammo.it

IL CAMPIONAMENTO E LA RIPRODUZIONE DEI MOVIMENTI

Supponiamo di volere insegnare al nostro piccolo figlio un nuovo lavoro manuale: probabilmente

Fig. 3: ***Il campionamento dei movimenti di un Robot, e la loro successiva riproduzione rappresentano un metodo molto efficiente di programmazione delle macchine per l'esecuzione di compiti ripetitivi.***

mostreremmo come eseguire la nuova azione, oppure prenderemmo la sua manina per mostrargli il movimento corretto. Dal momento che la 'manina' di un Robot generalmente è in grado di stritolare la nostra, appare appropriata la ricerca di un metodo differente di campionamento dei movimenti. Usualmente vengono utilizzati joystick per la definizione dei movimenti in una prima fase di apprendimento della macchina, durante la quale ogni singolo movimento viene registrato, generalmente a bassa velocità. La sequenza di movimenti così ottenuta è pronta per essere riprodotta, eventualmente ad una velocità maggiore, per un numero virtualmente infinito di volte. Nel caso di Robot antropomorfi è possibile utilizzare una serie di sensori posti in prossimità delle giunture dell'arto dell'operatore, appositamente studiati per rilevarne i movimenti. Il campionamento effettuato con questa tecnica appare molto naturale ed immediato. Nel nostro caso utilizzeremo uno strumento software che ci permette di comandare il Robot nella fase di apprendimento per mezzo di Trackbar (una per ogni attuatore), mentre i movimenti vengono campionati e registrati. In fase di esecuzione, la sequenza potrà essere ripetuta in modo ciclico, potrà essere interrotta in qualunque momento e si potranno inserire ulteriori sequenze di azioni. Nel caso pratico del nostro braccio meccanico, vogliamo implementare il controllo sincrono seriale dei primi due attuatori, con una risoluzione di otto bit ciascuno, permettendoci l'azionamento dei movimenti della flessione e della rotazione del polso. Anche questa volta non utilizzeremo computer industriali PLC: tutto questo è possibile con una manciata di componenti, che a stento raggiungono il costo di poche decine di euro.

SCHEDA UNIVERSALE DI OUTPUT PER ROBOTICA

Nell'articolo pubblicato nel numero precedente; è stato proposto un circuito universale di controllo seriale per robotica, che ha la caratteristica di poter comandare un servocomando a controllo di modulazione di lunghezza di impulso (PWM), nonché dotato di una uscita analogica e di otto uscite digitali. Per facilitare la realizzazione di nuove applicazioni, questo circuito è stato incluso in un'unica scheda universale, facilmente utilizzabile, versatile e pronta all'uso. È possibile collegare più schede in serie per moltiplicare il numero di servocomandi, di linee analogiche o digitali che si rendano necessarie, ed è inoltre possibile sviluppare nuove applicazioni che l'utente voglia implementare, semplicemente collegando i circuiti da comandare al connettore che è riportato in alto sulla scheda. Alcune applicazioni di questa scheda possono essere il controllo di motori passo-passo, la gestione di carichi elettrici attraverso il controllo digitale ed il comando di apparecchiature attraverso segnali analogici. Nel caso si voglia espandere un sistema, successivamente alla fase di progetto iniziale, sarà sufficiente aggiungere nuovi moduli ed applicare alcune minime modifiche al software di controllo. Lo sviluppo di nuove applicazioni con questo metodo appare veloce e potente ed ingloba tutti i vantaggi di un sistema modulare. La scheda universale di output è compatibile e direttamente collegabile ad altre schede che sono state progettate con la stessa filosofia e che verranno proposte successivamente. Nella applicazione che verrà esposta di seguito utilizzeremo una scheda universale di output, collegata ad un circuito gemello che verrà cablato su una apparecchiatura PC Explorer light: dimostreremo come sia possibile collegare questi due circuiti in modo tale da potere comandare due servomeccanismi PWM contemporaneamente.

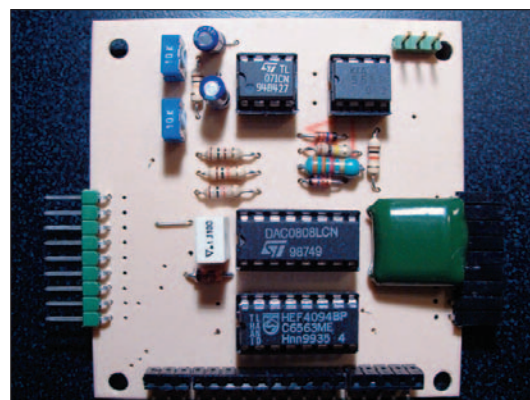



Fig. 4: ***La scheda universale di output è un componente modulare: per moltiplicarne le potenzialità è sufficiente collegarne il numero richiesto in serie.***

LO SCHEMA ELETTRICO

La struttura modulare che è stata scelta nello sviluppo dell'elettronica di questa applicazione permette l'utilizzo di due circuiti elettronici praticamente

NOTA

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo pcexplorer@elisys.it, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/495483.

BIBLIOGRAFIA

• **CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6** *Luca Spuntoni* ioProgrammo N. 57-58 Aprile e Maggio 2002.

• **ELETTRONICA E ROBOTICA** *Luca Spuntoni* ioProgrammo N. 71-72-73 Settembre, Ottobre e Novembre 2003.

identici, che hanno lo scopo di comandare due servocomandi indipendenti, che vengono controllati per mezzo di una sequenza di bit inviati serialmente. Il circuito elettrico base è stato già analizzato nel numero precedente di questa rivista: un circuito gemello è stato inserito in una scheda che viene collegata alla apparecchiatura PC Explorer light, per mezzo delle connessioni che vengono descritte di seguito: il lettore ha tutte le informazioni che sono necessarie alla costruzione di questa applicazione.
Analizzando lo schema elettrico riportato nella figura successiva, che per comodità del lettore è stato inserito nel CD allegato alla rivista all'interno del file *'Robot_Immagini.zip'*, insieme a tutte le fotografie relative ai cablaggi necessari alla realizzazione della apparecchiatura, notiamo che la struttura generale del circuito è molto simile a quella del circuito presentato nell'ultimo appuntamento, con l'aggiunta della scheda universale di output, inserita sul lato destro. Le poche connessioni necessarie ci fanno apprezzare immediatamente la potenza dell'approccio modulare. Scendendo nel dettaglio notiamo che la linea di CLOCK e quella di STROBE, relative rispettivamente al *Clock* (necessario al sincronismo per la trasmissione dei dati seriale sincrona) ed al segnale di memorizzazione della parola trasmessa all'interno della 'memoria' contenuta all'interno del circuito IC2. La linea *QS* proveniente dal circuito relativo al primo servocomando viene collegata all'ingresso DATA IN della scheda universale di output. Operando questi pochi collegamenti, oltre alle linee necessarie all'alimentazione dei circuiti, accade che in dato seriale inviato tramite la porta parallela all'ingresso DATA IN dell'integrato IC2 viene fatto traslare all'interno del registro a scorrimento in esso contenuto, fino a quando, dopo l'ottavo bit viene riportato in uscita sulla linea di riporto QS, che collegata all'ingresso DATA IN della scheda universale ne permette la memorizzazione nell'analogo registro a scorrimento qui contenuto. Terminata la trasmissione del sedicesimo bit, si opera l'invio del segnale di STROBE, che forza la memorizzazione del

Fig. 5: ***In figura viene riportato lo schema elettrico del circuito di controllo, che per comodità e su richiesta dei lettori è stato inserito all'interno del file: Robot_Immagini.zip, con il nome: DoppioDriver16Bit_Schema_Elettrico.bmp***

dato seriale contenuto nei registri a scorrimento all'interno dei circuiti di memoria LATCH inglobati sempre in IC2 per il successivo invio al convertitore Digitale-Analogico e per la generazione dei segnali di controllo dei servomeccanismi PWM. Sul lato sinistro dello schema si possono notare le connessioni alle linee di alimentazione dell'apparecchiatura *'PC Explorer light'*, sulla quale è possibile avere maggiori informazioni visitando il sito: *http://www.pcexplorer.it.*

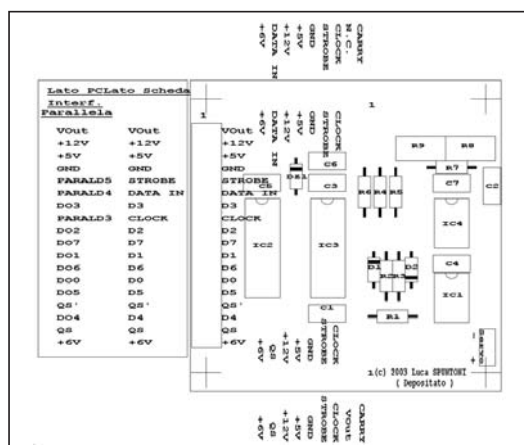
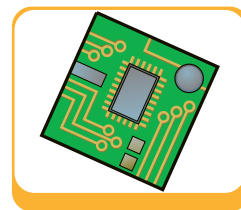



Fig. 6: ***L'immagine di figura raffigura le connessioni presenti sulla scheda universale di output, anche questa immagine è contenuta nel file Robot_Immagini.zip, con il nome: Connessioni_Scheda_Universale_Output.bmp.***

LA REALIZZAZIONE DEL CIRCUITO

La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file incluso al CD: *Robot_Immagini.zip*, con il nome: *DoppioDriver16Bit_Schema_Elettrico.bmp*.

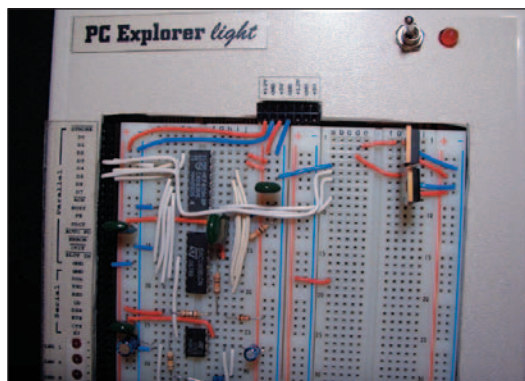



Fig. 7: ***Dettaglio del cablaggio nella parte alta del circuito Iniziando la realizzazione del circuito, predisponiamo i componenti sulla breadboard (qui viene utilizzato il sistema PC Explorer light).***

Per comodità del lettore, sono state incluse anche le immagini utili alla realizzazione della applicazione, per un più veloce ed immediato cablaggio dei cir-

IL BRACCIO MECCANICO PC EXPLORER LIGHT E LA SCHEDA UNIVERSALE DI OUTPUT

Con questo articolo concludiamo la descrizione del braccio meccanico, per i lettori che volessero maggiori informazioni sulla sua realizzazione e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: http://www.pcexplorer.it oppure http://web.tiscali.it/spuntosoft/, od infine scrivere all'indirizzo: spuntosoft@tiscali.it.

Fig. 8: *Dettaglio del cablaggio nella parte bassa del circuito.*

cuiti. Provvediamo ad inserire prima i componenti più piccoli ed a profilo più basso, ovvero i circuiti integrati, i diodi e le resistenze, posizioniamo quindi i condensatori. Le immagini riportate in queste pagine sono utili, insieme all'analisi dello schema elettrico ai fini della costruzione del circuito. Il cablaggio è stato realizzato utilizzando l'apparecchiatura PCExplorer light, in alternativa è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente. Una volta terminato il cablaggio del circuito è possibile inserire la scheda universale di output e provvedere a collegare i servocomandi PWM: l'applicazione a questo punto è completa dal punto di vista elettronico.

Fig. 9: *Il software di gestione permette il controllo di due servomotori, la registrazione dei movimenti del Robot e la loro riproduzione.*

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto. L'utilizzo del programma presentato in questa sede mentre è collegata una qualunque altra periferica al PC sulla porta LPT1, può bloccarne il funzionamento.

IL SOFTWARE DI CONTROLLO C++

Il software di controllo ha il compito primario di gestire contemporaneamente due servomeccanismi PWM; ha inoltre la capacità di registrare i movimenti che vengono fatti effettuare dall'operatore ed eventualmente è in grado di riprodurli in modo ciclico. Il programma è responsabile della gestione di tutta la applicazione: gli schemi elettrici infatti sono ridotti all'essenziale e deve essere posta la massima cura da parte del programmatore affinché condizioni proibite nello stato logico delle linee hardware di controllo non vengano a verificarsi. Il codice sorgente, scritto in C++ viene messo a completa disposizione del lettore ed è disponibile nel CD con il nome: *SpuntoRecorderTwinPWMServo_parallel.zip*: in questa sede analizziamo soltanto le parti più significative, rimanendo a disposizione del lettore per ogni chiarimento all'indirizzo: *luca.spuntoni@ioprogrammo.it*. Degna di nota, all'interno della procedura *FormCreate*, oltre all'inizializzazione dei vari parametri relativi alle porte fisiche e delle varie altre componenti del programma è l'inizializzazione dell'array all'interno del quale andranno memorizzate le word contenenti la codifica dei singoli movimenti del braccio meccanico: in particolare occorre notare che l'inizializzazione avviene impostando tutti gli elementi degli array al valore binario *'10000000 10000000'*, corrispondente al posizionamento di entrambi i servocomandi nella posizione centrale.

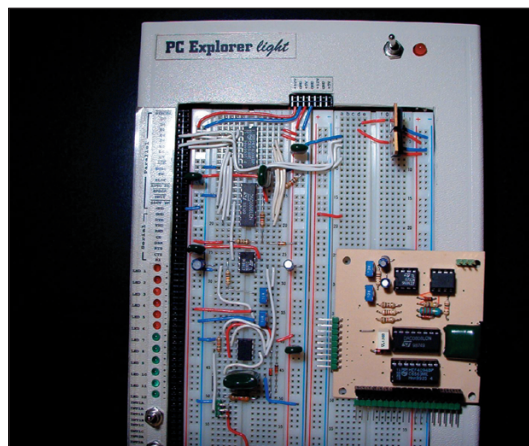
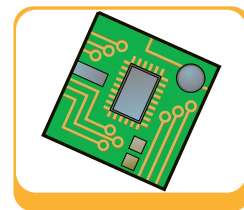



Fig. 10: *La connessione della scheda è semplice ed avviene attraverso l'apposito connettore.*

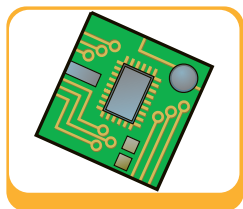
I componenti necessari al programma, sono forniti nello stesso file citato in precedenza e sono utilizzabili sia in C++ che in Delphi. Il cuore del programma risiede nella procedura *MainTimerTimer*, riportata di seguito, che il lettore potrà riconoscere come molto simile a quella analoga trattata nel numero precedente di questa rivista, dalla quale differisce però perché è in grado di inviare una parola di 16 bit, anziché di soltanto 8, per rendere possibile la gestione di due servocomandi.

```
//------------------------------------------------------------
void __fastcall TSpuntoRecorderTwinPWMServo16BitForm::
```

```
                    MainTimerTimer(TObject *Sender)
{  //Main Timer
   if (PowerONSpeedButton->Down)  //Power is ON
   {
    //*** DATA ***
    DelayTimer->Enabled=true;
    Bitout=(Dataout & 0x01); // Reads the most right bit
    // Updates the labels
    Label15->Caption = IntToStr(Contabit);
    Label16->Caption = IntToStr(Bitout);
    // Shifts the Output Byte Right 1 bit
    Contabit=Contabit + 1;
    Dataout=(Dataout >> 1);
  if (Bitout==0)  // Serial Data bit is 0
  {
  Datain=SpuntoHardwarePort->ReadPort(LPT1DataAddress);
  Datain=(Datain & 0xCF);      // Clock is 1 Serial Bit i
                                          s 0  Strobe is 0
  SpuntoHardwarePort->WritePort(
                          LPT1DataAddress,Datain);
  SerialDataSpuntoLed->LedOff();
  }
  else                    // Serial Data bit is 1
  {
  Datain=SpuntoHardwarePort->ReadPort(LPT1DataAddress);
  Datain=(Datain | 0x10);    // Clock is 1 Serial Bit is 1
  SpuntoHardwarePort->WritePort(
                          LPT1DataAddress,Datain);
  SerialDataSpuntoLed->LedOn();
  }
  //*** CLOCK ***
  SpuntoHardwarePort->LedOn();
  Datain=SpuntoHardwarePort->ReadPort(
                                 LPT1DataAddress);
  Datain=(Datain | 0x08);      // Clock bit to 1
  SpuntoHardwarePort->WritePort(
                          LPT1DataAddress,Datain);
  if (Contabit==16)       // If all 16 bits have been sent
  {
  //*** STROBE ***
  StrobeSpuntoLed->LedOn();
  Datain=SpuntoHardwarePort->ReadPort(
                                 LPT1DataAddress);
  Datain=(Datain | 0x20);  //  Strobes the Byte to
                        the 4094 Latch  (Strobe is 1)
  SpuntoHardwarePort->WritePort(
                          LPT1DataAddress,Datain);
  // Bit counter reset
  Contabit=0;  // Resets the bit position
  Dataout=PWMTrackbar1->Position +
                      (256*PWMTrackbar2->Position);
  // Reads the Trackbar position for the next conversion
```

La procedura *MainTimerTimer* provvede a generare i segnali di CLOCK, necessario al sincronismo della trasmissione seriale, DATA contenente il dato seriale da trasmettere e STROBE, che viene inviato in corrispondenza del sedicesimo bit per consentire la memorizzazione dell'intera word trasmessa serialmente all'interno del convertitore seriale-parallelo IC2. Al termine di ciascun impulso vengono posti a livello logico basso le tre linee di CLOCK, DATI e STROBE: La parte di codice deputata alla gestione del nostro 'registratore' di movimenti del braccio meccanico, riportata di seguito, è contenuta sempre nella procedura *MainTimerTimer*, attivata ogni qualvolta il timer *MainTimer* induce l'esecuzione di questo event handler. La sequenza di movimenti viene registrata in un array, inizializzato nella procedura *FormCreate* (e poi distrutto in *FormDestroy*), all'interno del quale viene memorizzata la sequenza di 16 bit trasmessa all'elettronica di controllo, al momento dell'invio del comando di azionamento degli attuatori, corrispondente al segnale di STROBE della memoria LATCH contenuta nel circuito integrato IC2.

```
 //*** Recorder and Player ***
 if  (SpeedButtonRec->Down==True)
                  ArrayDataOut[PresentRecord]=Dataout;
                        // Records the Servo position
 if  (SpeedButtonPlay->Down==True) Dataout=
                       ArrayDataOut[PresentRecord];
                          // Plays the Servo position
   if ((SpeedButtonPlay->Down==True) ||
                      (SpeedButtonRec->Down==True))
   { PresentRecord++; }
 }  // If (Contabit==16) ends here but the Recorder
                      and Player section continues
 PresentRecordLabel->Caption = IntToStr(PresentRecord);
 if ((SpeedButtonPlay->Down) &&
     (PresentRecord>LastRecord)) PresentRecord=0;
 if (PresentRecord==MaxDataOutRecord)
                                  PresentRecord=0;
 if ((SpeedButtonRec->Down==True) &&
                      (PresentRecord>LastRecord))
                           LastRecord=PresentRecord;
}
else
 {
 SpuntoHardwarePort->LedOff(); // Power is OFF
 DelayTimer->Enabled=false;
 }
}
//----------------------------------------------------------
void __fastcall
        TSpuntoRecorderTwinPWMServo16BitForm::
                   DelayTimerTimer(TObject *Sender)
{   //Delay timer
 Datain=SpuntoHardwarePort->ReadPort(
                                 LPT1DataAddress);
 Datain=(Datain & 0xC7);  // Clock is 0 Serial Bit is 0
                                        Strobe is 0
 SpuntoHardwarePort->WritePort(
                          LPT1DataAddress,Datain);
 SpuntoHardwarePort->LedOff();
```

IL SOFTWARE

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD allegato alla rivista all'interno del file: 'SpuntoRecorderTwinPWMServo_parallel.zip'. Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, oppure NT, è possibile utilizzare un driver, quale 'PortTalk' (PortTalk22.zip), scaricabile del sito: http://www.beyondlogic.org.

```
SerialDataSpuntoLed->LedOff();
StrobeSpuntoLed->LedOff();
DelayTimer->Enabled=false; }
//-----------------------------------------------------------
```

Il programma ha la caratteristica di accedere all'hardware del PC attraverso i propri indirizzi fisici di I/O, questa tecnica, dal momento che scavalca il

Fig. 11: *La connessione di una scheda universale di output permette di espandere l'applicazione gestendo un servocomando aggiuntivo.*

sistema operativo, potrebbe non 'piacere' a Windows NT, 2000, oppure XP, per ovviare a questo problema, se non si vogliono utilizzare sistemi operativi 'datati' come Win 9.X, o Me è possibile , per non incorrere ad un errore del tipo *"Privileged error"*, utilizzare un driver, quale PortTalk (*PortTalk22.zip*), scaricabile del sito: *www.beyondlogic.org*. Il software a questo punto è completo ed è in grado di gestire indipendentemente due servocomandi, di registrarne la sequenza di posizionamento e di riprodurne i movimenti.

COLLAUDO DEL SISTEMA

A questo punto siamo finalmente pronti per fare mescolare il caffè al nostro braccio meccanico. Innanzi tutto prima di collegare il circuito al nostro PC occorre verificare la realizzazione con attenzione per assicurarci che tutto sia stato connesso come previsto: controlliamo che i connettori siano ben serrati e che nessuna parte metallica della mano possa urtare il circuito elettrico. Colleghiamo al nostro circuito il cavo relativo alla porta parallela del PC, se possediamo *'PC Explorer'* oppure *'PC Explorer light'* come mostrato in figura, oppure provvedendo a costruire un cavo seguendo lo schema elettrico e la tabella riportati all'inizio dell'articolo. Posizioniamo la tazza di caffè sotto il braccio meccanico, lanciamo il programma di gestione, alimentiamo il circuito e utilizzando le Trackbar spostiamo il braccio (per ora solo su due dei sei assi che avrà in futuro) fino a posizionare il cucchiaino nella tazzina e registriamo il movimento più idoneo a mescolare il caffè come più ci aggrada. Fermando la registrazione e premendo il tasto play dovremmo vedere il braccio meccanico ripetere fedelmente i movimenti appena registrati, inoltre è possibile variare la velocità di esecuzione, semplicemente variando il periodo relativo alla trasmissione dati.

Se il circuito non funziona, provvediamo a spegnere tutto prima di ricontrollare i collegamenti e riprovare di nuovo. Siamo in grado a questo punto di muovere il nostro braccio meccanico con precisione: sostituendo alla trackbar della posizione un algoritmo di controllo tridimensionale, siamo pronti ad entrare nel mondo dell'automazione e della robotica.

Fig. 12: *Su www.ioprogrammo.it alla rivista è disponibile un filmato che mostra la mano meccanica in azione con nostra tazzina di caffè (Robot_Caffe.AVI).*

SUL WEB

Il sistema proposto in queste pagine è stato realizzato e collaudato con la apparecchiatura per il collaudo e la sperimentazione di circuiti elettronici con Personal Computer 'PC EXPLORER light'. Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: http://www.pcexplorer.it

CONCLUSIONI

In queste pagine abbiamo presentato la prima di una lunga serie di applicazioni di robotica, che ci ha fornito la capacità di muovere i primi due assi il braccio meccanico, registrarne i movimenti ed eventualmente riprodurli. Abbiamo verificato la capacità della applicazione di interagire con una tazza di caffè con sufficiente precisione in modo tale da mescolarne il contenuto senza rovesciarlo. Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore.

Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.

Luca Spuntoni
luca.spuntoni@ioprogrammo.it

La programmazione hardware in VB

Controllare un dispositivo USB da Visual Basic

La tecnologia USB ha ormai "invaso" tutti i moderni personal computer e non solo. Ecco perché nasce l'esigenza di un articolo che spieghi come sviluppare applicazioni in grado di interagire con questo dispositivo hardware.

I dispositivi USB *(Universal Serial Bus)* nascono per superare le limitazioni dei precedenti standard in merito al numero dei dispositivi collegabili contemporaneamente ad un PC, tramite collegamento in cascata dei dispositivi. Più precisamente, la versione 2.0 delle specifiche dello standard (reperibile presso il sito di riferimento *http://www.usb.org*) prevede una serie di caratteristiche: possibilità d'inserimento "a caldo" di una periferica (il software di inizializzazione USB è sempre attivo, e non solo quando il PC è in funzione), facilità d'uso, standardizzazione dei connettori, alte prestazioni (lo standard attuale prevede tre tipi di collegamento, rispettivamente a 1.5, 12 e 480 Mbps), possibilità di connettere fino a 126 dispositivi in cascata, alimentazione fornita dal cavo (fino ad un limite di 500 mA) e gestita in maniera "intelligente" (un dispositivo USB va automaticamente in stand-by quando non è utilizzato), auto-discovery degli errori e capacità di auto-recovery, possibilità di operare senza necessità di aprire il case, onde installare schede addizionali. Nella pratica, USB consiste di un protocollo seriale e di un link fisico, che trasmette i dati in maniera differenziale su una coppia di cavi, mentre un'altra coppia trasmette potenza ai dispositivi collegati in cascata (Fig. 1). È stata adottata la comunicazione seriale, in luogo di quella parallela, in quanto si voleva che i cavi fossero i più semplici possibile, in modo da poter implementare con facilità configurazioni dinamiche. Poiché un cavo USB presenta due connettori fisicamente diversi, non è possibile installare un cavo in maniera scorretta.
Quando una periferica viene collegata alla rete USB, l'host instaura una comunicazione con il dispositivo per conoscerne l'identità e stabilire quale tipo di driver è richiesto: questo processo è la cosiddetta *enumeration dei dispositivi.*
Notiamo, sempre in Fig. 1, la presenza dei cosiddetti *hub,* utilizzati per incrementare il numero dei dispositivi collegati ad un singolo host; più precisamente, un hub ha un'unica connessione di tipo upstream, verso il PC host o un altro hub, ed una o più connessioni di tipo downstream verso altri nodi della rete USB.

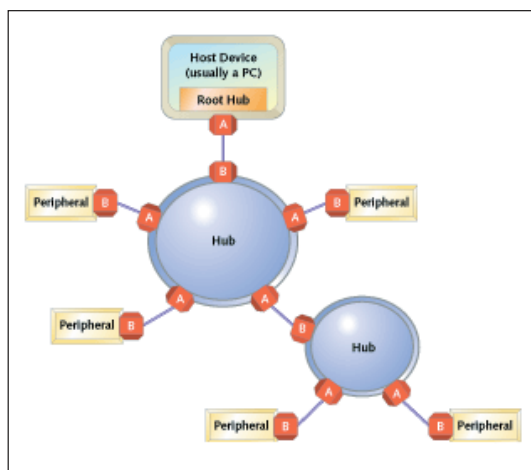
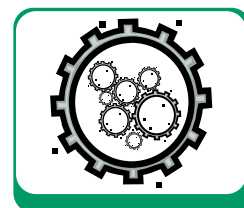



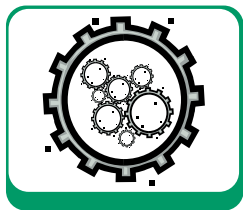
***Fig. 1:* La topologia di una rete di collegamenti USB.**

INIZIAMO CON UN PO' DI TEORIA

La comunicazione USB avviene fra il PC host ed i "punti terminali" (*endpoints*) situati in una periferica. Un punto terminale è definito come una porzione, cui viene assegnato un indirizzo unico, della periferica, atta ad inviare o ricevere dati. Quattro bit definiscono l'indirizzo di un punto terminale; questa codifica indica, inoltre, la direzione di trasferimento dati e se vengono trasmesse informazioni di controllo. L'indirizzo 0 è riservato alle comunicazioni di controllo, lasciando così 15 punti terminali per ogni dispositivo, che possono fungere da destinazioni o da sorgenti di dati. Il trasferimento dei dati avviene lungo delle pipe virtuali fra i punti terminali ed il PC host. Allorquando si stabilisce una comu-

SUL WEB

Informazioni sulla classe HID sul sito web USB.org
http://www.usb.org/developers/hidpage.html

USB Central
http://www.lvr.com/usb.htm

nicazione con la periferica, ogni punto terminale restituisce un descrittore, ovvero una struttura dati che "informa" il PC host circa la configurazione ed il ruolo dei punti terminali. Più precisamente, un descrittore comprende il tipo di trasferimento, la grandezza dei pacchetti di dati e, opzionalmente, l'intervallo d'attesa per il trasferimento dei dati e la larghezza di banda necessaria. Una volta ricevuto il descrittore, il PC host apre una pipe virtuale, riservando ad essa una certa larghezza di banda. USB supporta quattro tipi di trasferimento dati: di controllo, isocroni, di tipo *bulk*, ed *interrupt*. I dati di controllo comprendono quelli necessari a scambiare informazioni di configurazione, comandi e setup fra l'host e la periferica; il trasferimento è di tipo affidabile: i dati ricevuti vengono sottoposti ad un test CRC *(Cyclic Redundance Code)*; in sostanza, si sa che i bit di un pacchetto dati corretto presentano determinate caratteristiche per cui, se si fallisce un test CRC, il pacchetto viene scartato e ne viene richiesta

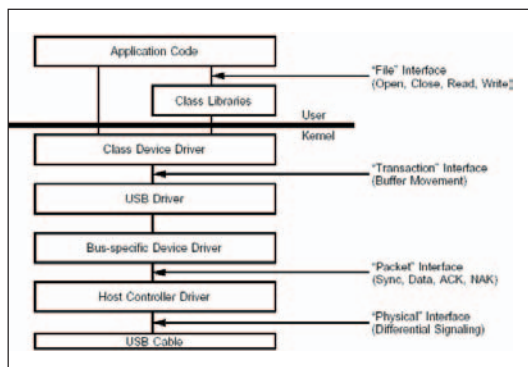



Fig. 2: ***il software del PC host dedicato ad USB è suddiviso in strati.***

la ritrasmissione. D'altronde, un pacchetto potrebbe essere corrotto in maniera tale da superare un test CRC; è cura però dello standard fare in modo che questa evenienza sia statisticamente irrilevante. Un trasferimento dati di tipo *bulk* si verifica allorquando vengono scambiate grandi quantità di dati ed uno scambio tempestivo non è critico: ad esempio, quando si comunica con stampanti e scanner. I trasferimenti di tipo interrupt sono in un certo senso complementari a quelli di tipo bulk: si tratta, in effetti, di piccole quantità di dati che richiedono immediata attenzione, e sono caratteristici delle comunicazioni con mouse e tastiere. Infine, i trasferimenti isocroni sono caratteristici degli streaming continui come quelli verso dispositivi audio/video; a causa dello scambio che avviene in tempo reale, questi trasferimenti sono gli unici non sottoposti a test CRC. Veniamo all'architettura software di supporto ad USB. USB è uno standard complesso, che richiede un enorme supporto software, sia in termini di firmware del dispositivo che sul PC host. Ci focalizzeremo sul mondo Microsoft. USB non è supportato, neanche in minima parte, in MS-DOS, Windows 3.x e Windows NT. Com'è probabilmente noto, la Microsoft ha iniziato a supportare USB con le ultime versioni (a partire dalla OEM 2.1) di Windows 95, riservando la piena implementazione dello standard a Windows 98. Ora, si può affermare con tranquillità che sviluppare un driver USB non è compito alla portata della maggior parte dei programmatori. La "buona novella" è che, nella maggior parte dei casi, i driver inclusi in Windows andranno più che bene per la nostra periferica. Diamo, perciò, un'occhiata più da vicino al funzionamento dei driver Windows (Fig. 2 e 3).

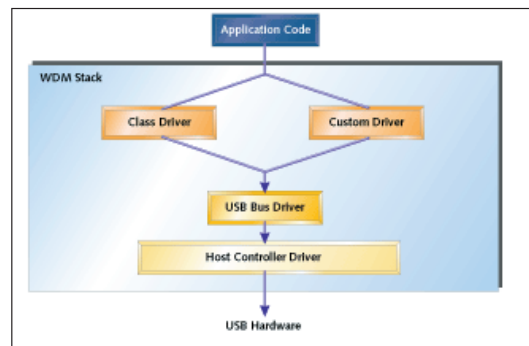



Fig. 3: ***L'implementazione Windows dello stack USB, aderente al modello Windows WDM (Win32 Driver Model).***

A partire da Windows 98, i driver Microsoft presentano una struttura caratteristica, nota come *Win32 Driver Model* ovvero *WDM* (Fig. 3), la quale suddivide le differenti parti del processo di comunicazione in uno stack di driver. Il codice applicativo (tramite le chiamate alle API di Windows) comunica con i driver custom o di classe del WDM (vedremo tra poco cos'è una "classe"). All'interno dello stack WDM il trasferimento dei dati utilizza pacchetti di basso livello, gli IRP *(I/O request packets)*, piuttosto che chiamate alle API. L'*USB Bus Driver* gestisce l'alimentazione ai dispositivi USB, il processo di enumeration, ed in genere le varie transazioni USB. Al di sotto di esso, l'Host Controller dialoga direttamente con l'hardware USB. Gli ultimi due driver vengono forniti direttamente con Windows, per cui il programmatore viene sgravato dal compito di svilupparli. Windows, così come le specifiche USB, suddivide i driver in "classi"; i dispositivi all'interno della stessa classe condividono un set comune di interfacce. Un esempio è la classe dei dispositivi di interfaccia uomo-macchina (human interface device o, nel seguito, HID), che comprende mouse, joystick e tastiere. Windows, a partire dal 98, viene rilasciato con un set completo di driver per la classe HID, pienamente rispondente alle specifiche USB: ne consegue che, se una periferiche richiede software di supporto che cade all'interno di questa classe, è possibile utilizzare i driver di Windows senza necessità di scrivere codice lato host. Nel seguito, pertanto, focalizzeremo l'attenzione sui dispositivi di classe HID.

In alternativa ai driver di classe è, naturalmente, possibile scrivere dei driver custom (vedi Fig. 2 e 3), ad esempio se abbiamo un sistema di acquisizione dati non standard. Visual C++, è quasi superfluo dirlo, è in grado di compilare i driver WDM: allo scopo, è necessario scaricare il *Driver Developer's Kit* (DDK; quello per Windows 98 è disponibile all'indirizzo *www.microsoft.com/DDK/ddk98.htm*), che comprende anche numerosi esempi di codice. La Blue Water Systems (*www.bluewatersystems.com*) fornisce anch'essa un kit di sviluppo; si faccia attenzione a scaricare anche l'ottimo USB Extensions Toolkit. Per scambiare dati con un dispositivo USB il codice applicativo farà riferimento ad esso come se si trattasse di un file, utilizzando un handle standard di Windows per identificare il dispositivo.

COME GESTIRE UN DISPOSITIVO HID

Il testo della Intel *"USB Design By Example, Second Edition"* (vedi box laterale) comprende delle applicazioni di esempio (fornite a corredo e presenti nel file *USBCod.zip*) che permettono di controllare un generico dispositivo USB utilizzando il supporto alla classe HID già inclusa in Windows (Fig. 4).

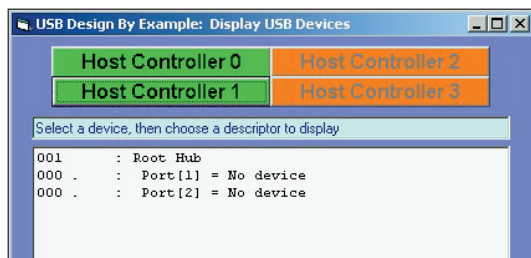



Fig. 4: ***L'applicazione displayusb, disponibile nei CD-Rom allegato o sul Web, permette di controllare un generico dispositivo USB utilizzando il supporto alla classe HID già inclusa in Windows.***

Insieme a questa utility può essere utilizzata con profitto quella inclusa nel file *dt2_4.zip* in allegato, il *Descriptor Tool 2.4* (Fig. 5), disponibile sul sito di riferimento USB Central; quest'ultima consente di ottenere il descrittore di uno specifico dispositivo USB. Infine, l'utility *usbhidio*, anch'essa disponibile su USB Central e disponibile nel file *usbhidio*, permette di scambiare dati con una periferica USB (Fig. 6). Riesaminiamo il codice d'esempio del testo della Intel. Il codice di più generale interesse è, sicuramente, quello contenuto nelle librerie comuni incluse nel file *Commvb.zip*. In particolare, il modulo *OSInterface.bas* dichiara i tipi di dato e le chiamate di sistema necessarie per accedere al sistema operativo Windows; si noti quante delle funzioni sono prese dalla libreria *HID.dll*, così come le funzioni di gestione degli handle associati ai dispositivi USB sono, come ci aspettavamo, quelle fornite dalle librerie di sistema per la gestione dei file. Il modulo *HIDInterface.bas* contiene, invece, le chiamate necessarie per le operazioni di base (*Open, Read, Write, Close*) su un dispositivo USB. A titolo di esempio, notiamo che fa la funzione *OpenUSBdevice*, il cui scopo è quello di scorrere la tabella di sistema dei dispositivi HID alla ricerca del nome di dispositivo passato come argomento, se lo trova, apre il dispositivo e restituisce *TRUE*, altrimenti ritorna *FALSE*. In primo luogo, la funzione ottiene l'identificatore di classe *HID*; la successiva operazione è quella di ottenere l'handle del dispositivo, e l'elenco dei dispositivi HID correntemente attivi. Viene poi effettuata la ricerca nella tabella dei dispositivi HID. Se il dispositivo viene identificato, è necessario convertire il nome, ritornato come una stringa C, in una stringa Visual Basic. È ora possibile "aprire" il dispositivo: viene creato un handle per esso (si noti l'invocazione alla funzione *CreateFile()*), e ci si accerta che si tratti proprio del dispositivo da noi cercato, nel qual caso lo si apre e si fa ritornare *TRUE* alla funzione; altrimenti, è necessario chiuderlo con una *CloseHandle()* prima di proseguire con lo scorrimento della tabella dei dispositivi HID. Prima di uscire dalla funzione, la *SetupDiDestroyDeviceInfoList()* (appartenente alla libreria *setupapi.dll*) cancella la lista dei dispositivi HID attivi.

Paolo De Nictolis

Fig. 5: ***Il Descriptor Tool 2.4 permette di ottenere i descrittori generati da una periferica USB.***

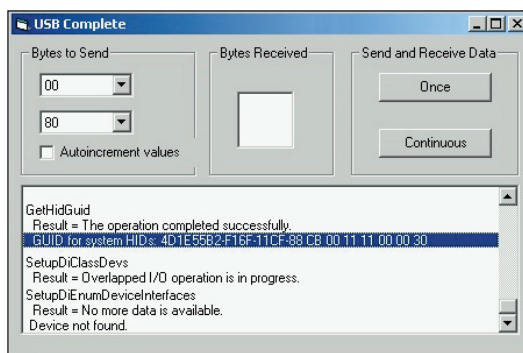
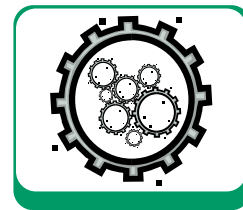



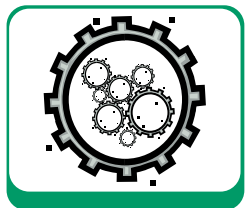
Fig. 6: ***L'utility usbhidio permette di scambiare dati con una periferica USB.***

- **USB DESIGN BY EXAMPLE, SECOND EDITION**
  John Hyde
  (Intel Press)
  http://www.intel.com/intelpress/usb/

- **AN INTRODUCTION TO USB DEVELOPMENT**
  Jack G. Ganssle
  http://www.embedded.com/internet/0003/0003ia2.htm

Integrare un motore di help in applicazioni Java

Creare l'help online in Java

parte seconda

Nell'articolo precedente abbiamo illustrato JavaHelp, le sue potenzialità e i vari scenari di utilizzazione possibili. In questo appuntamento, sperimenteremo alcune personalizzazioni.

Abbiamo creato una struttura di help senza scrivere neanche una riga di codice Java. In questo articolo vedremo come personalizzare ulteriormente il nostro help, effettuare l'unione tra diversi helpset ed inizieremo ad esplorare le potenzialità di tali API all'interno di applicazioni standalone.

FINESTRE SECONDARIE E DI POPUP

È possibile inserire all'interno del nostro help delle finestre secondarie o di popup in cui inserire delle informazioni aggiuntive. Per definire tali finestre occorre utilizzare il tag *<object>*. I parametri principali sono:

- **viewerStyle**: Definisce il tipo di finestra. I valori possibili sono *javax.help.Popup* o *Javax.help.SecondaryWindow.*
- **viewerActivator**: Definisce il tipo di oggetto su cui l'utente dovrà cliccare per attivare la finestra. I valori possibili sono: *javax.help.LinkButton* o *javax.help.LinkLabel*
- **viewerLocation**: Posizione del pulsante
- **text:** Testo da aggiungere al pulsante

Fig. 1: *Finestra secondaria.*

- **content:** Topic da aprire
- **viewerSize:** Dimensione del pulsante

Una finestra secondaria (Fig. 1) è costituita da un singolo pannello e visualizza le informazioni di help dell'argomento scelto. È simile al content pane, può essere ridimensionata e spostata dall'utente ma, a differenza di quest'ultimo, se una finestra secondaria viene chiusa verrà distrutta.

```
<OBJECT CLASSID="java:com.sun.java.help.impl.
                                    JHSecondaryViewer">
<param name="content" value="fileDiContenuto.htm">
<param name="viewerActivator" value=
                                  "javax.help.LinkLabel">
<param name="viewerStyle" value=
                          "javax.help.SecondaryWindow ">
<param name="viewerSize" value="400,250">
<param name="text" value="  Secondary Window">
<param name="textColor" value="bllue">
</OBJECT>
```

La finestra di popup (Fig. 2) ha solo un pannello di tipo content viewer e resta aperta solo finché ha il focus, appena lo perde viene distrutta. Non può essere ridimensionata o spostata dall'utente.

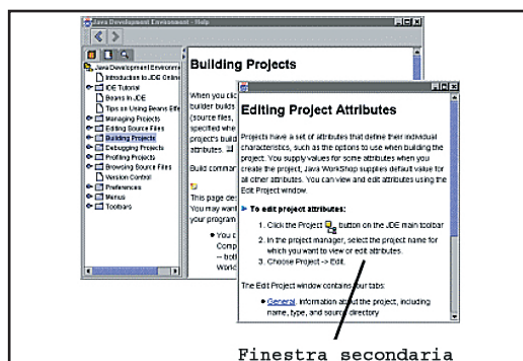



Fig. 2: *Finestra di popup.*

```
<OBJECT CLASSID="java:com.sun.java.help.impl
                                  .JHSecondaryViewer">
<param name="content" value="fileDiContenuto.htm">
<param name="viewerActivator" value=
                                  "javax.help.LinkLabel">
<param name="viewerStyle" value="javax.help.Popup">
<param name="viewerSize" value="400,250">
<param name="text" value="Finestra di PopUP  ">
<param name="textColor" value="blue">
<param name="viewerName" value="glossary">
</OBJECT>
```

ICONE PERSONALIZZATE

Un ultima personalizzazione consiste nel modificare le icone utilizzate nelle finestre navigazionali (Es. la TOC). Per fare ciò occorre specificare le immagini come se fossero dei topic nel map file:

```
<mapID target="toplevelfolder" url="images/toplevel.gif"/>
<mapID target="chapter" url="images/chapTopic.gif"/>
<mapID target="topic" url="images/topic.gif"/>
```

Successivamente, nel file TOC, utilizzando gli attributi *categoryclosedimage* e *topicimage*, specifichiamo le immagini, per i capitoli e per i topic:

```
<toc categoryclosedimage="chapter" topicimage="topic">
```

È possibile anche specificare immagini diverse per ogni. Nell'esempio successivo specifichiamo un'immagine differente per il primo item dell'albero:

```
<tocitem text="History of the Holidays" image=
                                  "toplevelfolder">
```

Merge - JavaHelp fornisce un meccanismo che consente di effettuare il merge, ovvero l'unione, di più helpset in uno solo. In realtà, viene effettuato il merge delle viste (TOC, Index, Glossary, ecc..) di uno o più helpset in un helpset già esistente chiamato master. Quando utilizzare il merge ? Tale funzionalità è utile per le suite di applicazioni composte da un insieme di software minori. Pensiamo a Microsoft Office costituito da diversi software: Word, Excel, Powerpoint etc. Ogni software della suite può essere installato o meno; se è installato, l'help relativo potrebbe essere incluso con le informazioni di help delle altre applicazioni della suite. Se non installo uno di questi software non ho la necessità dell'help corrispondente. Il merge può essere effettuato staticamente, specificando gli helpset da concatenare nell'helpset file (*.hs*) dell'helpset master, o dinamicamente, mediante un programma che utilizza le API di JavaHelp. Ci sono quattro tipi di merge che possono essere realizzati (Tab. 1).

Tipo di merge	Classe
SortMerge	*javax.help.SortMerge*
UniteAppendMerge	*javax.help.UniteAppendMerge*
AppendMerge	*javax.help.AppendMerge*
NoMerge	*javax.help.NoMerge*

Tabella 1:

Ogni tipo di vista ha la propria modalità di merge di default che è quella che verrà utilizzata se non ne viene indicata una differente. La modalità di unione per ogni singola vista può essere impostata utilizzando l'attributo mergetype del tag *<view>* definito nel file *.hs* dell'helpset master. Per esempio:

```
<view mergetype="javax.help.SortMerge">
<name>Index</name>
<label>Index</label>
<type>javax.help.IndexView</type>
<data>AnimalsIndex.xml</data>
</view>
```

imposta la modalità SortMerge per l'indice.

UniteAppendMerge - Questo tipo di merge preserva la gerarchia della vista del master effettuando il merge di elementi che si trovano allo stesso livello in un solo elemento e poi effettua il merge e l'ordinamento dei suoi sotto elementi. È utilizzato nelle viste di tipo TOC. Due elementi sono uniti in uno stesso elemento soltanto se sono identici, ovvero stesso nome e stesso ID al file di contenuto.

Master TOC	Vertebrates TOC	Invertebrates TOC	Merged TOC
Animal Categories	Animal Categories	Animal Categories	Animal Categories
Vertebrates	Vertebrates	Invertebrates	Vertebrates
Invertebrates	Fish	Protozoa	Fish
	Amphibians	Echnioderms	Amphibians
	Reptiles	Annilids	Reptiles
	Birds	Mollusks	Birds
	Mammals	Arthropods	Mammals
	Marsupials	Crustaceans	Marsupials
	Primates	Arachnids	Primates
	Rodents	Insects	Rodents
	Cetaceans	Pictures [Sort merge]	Cetaceans
	Animals Like Seals	Butterfly	Animals Like Seals
	Pictures [Sort merge]	Clam	Invertebrates
	Bat	Crab	Protozoa
	Bears	Dragon	Echnioderms
	Black Bear	Sea Star	Annilids
	Grizzly	Spider	Mollusks
	Koala	Sponge	Arthropods
	Bird	Worms	Crustaceans
	Crocodile		Arachnids
	Dolphin		Insects
	Elephant		Pictures
			Bat
			Bears
			Black Bear
			Grizzly
			Koala
			Bird
			Butterfly
			Clam
			Crab
			Crocodile

Fig. 3: *Esempio di SortMerge.*

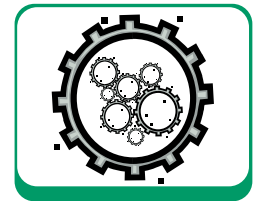
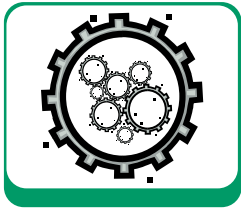
SortMerge - I dati delle viste vengono uniti secondo l'ordinamento alfabetico. Se vi è un entry nel master helpset che ha lo stesso nome e ID di uno presente nell'helpset da unire, quest'ultimo viene ignorato e verrà inserito soltanto l'entry del master helpset. Se soltanto i nomi sono identici, verranno inseriti entrambi, ma accanto al nome viene aggiunto, tra parentesi, il titolo dell'helpset dell'entry. Tale tipo di merge non dovrebbe essere utilizzato quando abbiamo una gerarchia, ma soltanto quando abbiamo informazioni che possono essere collegate come gli elementi di un indice o di un glossario.

Il content-pane è il pannello dei contenuti, ovvero il pannello in cui verrà visualizzata la pagina html contenente le informazioni di help.

Java Workshop Index	Java Studio Index	Merged Index
Menus Build Menu Debug Menu Edit Menu File Menu Help Menu Toolbars Edit/Debug Toolbar Main Toolbar	Developer Resources Examples List of Additional Examples Step-by-step Example Menus Edit Menu File Menu Help Menu View Menu Toolbars Composition Toolbar Main Toolbar	Developer Resources Examples List of Additional Examples Step-by-step Example Menus Build Menu Debug Menu Edit Menu (Java Workshop) Edit Menu (Java Studio) File Menu (Java Workshop) File Menu (Java Studio) Help Menu (Java Workshop) Help Menu (Java Studio) View Menu Toolbars Composition Toolbar Edit/Debug Toolbar Main Toolbar (Java Workshop) Main Toolbar (Java Studio)

Fig. 4: ***Esempio di UniteAppendMerge.***

AppendMerge - Appende i dati della nuova vista alla fine dei dati della vista master.
NoMerge - Non è effettuata nessuna elaborazione. La vista dell'helpset unito non appare nell'helpset finale.

MERGE STATICO

Si è già specificato che il merge di un helpset viene sempre effettuato all'interno di un altro helpset, chiamato master. Il master helpset può essere un helpset che presenta già dei dati o uno vuoto (dataless helpset), ovvero definisce una serie di viste vuote. Il dataless master helpset rappresenta un contenitore in cui vengono inclusi gli altri helpset.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
  <helpset version="2.0">
<!-- title -->
<title>JavaHelp test</title>
<!-- views -->
<view>
  <name>TOC</name>
  <label>Table Of Contents</label>
  <type>javax.help.TOCView</type>
</view>
<view>
  <name>Index</name>
  <label>Index</label>
  <type>javax.help.IndexView</type>
</view>
<view>
  <name>Search</name>
  <label>Search</label>
  <type>javax.help.SearchView</type>
</view>
<subhelpset location="app1.hs" />
<subhelpset location="app2.hs" />
<subhelpset location="app3.hs" />
<subhelpset location="app4.hs" />
  </helpset>
```

L'helpset contenente il tag <subhelpset> è considerato il master.

NOTA

HELPBROKER

L'HelpBroker è un agente che gestisce l'aspetto dell'help e fornisce i metodi per implementare l'help contestuale e tutte le funzionalità dell'help.

HELP ON LINE ALLE APPLICAZIONI STANDALONE

Dopo aver creato l'help, con tutti i file necessari e dopo averlo personalizzato, non ci resta che imparare come sia possibile richiamarlo da un'applicazione Java standalone. Le operazioni da compiere sono molto semplici:

1. **Importare il package javax.help:**

```
import javax.help.*;
```

2. **Trovare l'helpset file e creare un oggetto Helpset**

```
String helpset = "myHelpSet.hs";
HelpSet hs;
HelpBroker hb;
ClassLoader loader = this.getClass().getClassLoader();
try {
  URL hsUrl = HelpSet.findHelpSet(loader, helpset);
  hs = new HelpSet(loader, hsUrl);
}catch (Exception ee) {
 System.out.println("HelpSet " + helpset + " not found")
 return; }
```

3. **Creare un oggetto HelpBroker**

```
hb = hs.createHelpBroker();
```

4. **Creare un menù di Help**

```
……..
JMenu help = new JMenu("Help");
menuBar.add(help);
menu_help = new JMenuItem("Help");
menuhelp.addActionListener(new
                      CSH.DisplayHelpFromSource(hb));
```

Quando viene attivato un help il primo file che viene letto dall'applicazione e l'helpset file (*.hs*). L'HelpBroker è un'astrazione della rappresentazione di un HelpSet. Esso è un agente che gestisce l'aspetto dell'help e fornisce i metodi per implementare l'help contestuale e tutte le funzionalità dell'help.

COME RICHIAMARE L'HELP?

È possibile utilizzare diversi metodi per invocare l'help dall'interno di un programma. Il primo metodo, già illustrato precedentemente, consiste nel creare una voce di menù (*MenuItem*) e registrare come action listener una nuova istanza della classe *CSH.DisplayHelpFromSource*:

```
JMenu help = new JMenu("Help");
menuBar.add(help);
menu_help = new JMenuItem("Help");
menuhelp.addActionListener(new
                    CSH.DisplayHelpFromSource(hb));
```

In alternativa è possibile impostare l'avvio dell'help dopo la pressione di un pulsante:

```
Jbutton button = new Jbutton("Help");
hb.enableHelpOnButton(button, "mainID", null);
```

il metodo *enableHelpOnButton* registra l'*helpBroker* come ascoltatore degli eventi del pulsante. Se questi viene premuto avvia l'help e mostra il file di contenuto definito da *mainID*.
Ma qual è il metodo classico per richiamare un help in molti sistemi?
Premere il tasto *F1*. Anche JavaHelp può essere configurato per aprire l'help alla pressione di tale tasto. Basterà aggiungere la seguente linea di codice:

```
hb.enableHelpKey(rootPane, "mainID", null )
```

i parametri da passare sono il rootPane del Frame dell'applicazione e l'help ID dell'argomento da visualizzare (tipicamente quello principale). La figura 5 mostra un semplicissimo demo che illustra i diversi modi di richiamare l'help. Premendo il tasto *Run Help*, oppure tramite il menù *Help* e la voce *Contents* o tramite la pressione del tasto *F1*. Troverete tale applicazione all'interno del CD di ioProgrammo.

HELP CONTESTUALE

Una delle potenzialità di JavaHelp consiste nella possibilità di fornire un help contestuale, cioè è possibile richiedere direttamente le informazioni di help di un componente dell'interfaccia grafica del nostro programma o di una item di un menù.
La gestione dell'help contestuale è affidata alla classe CSH (Context Sensitive Help) che fornisce diversi metodi e le seguenti tre inner-class :

DisplayHelpFromFocus: localizza il componente che ha attualmente il focus, poi ricerca l'ID ad esso assegnato e lo visualizza nell'HelpBroker.
DisplayHelpAfterTracking: Attende che il mouse venga premuto per selezionare un componente, poi trova l'ID associato e lo presenta.
DisplayHelpFromSource: Trova l'ID assegnato alla sorgente dell'action event e lo presenta.
E' utilizzata per avviare l'help tramite un pulsante o una voce di menù.

Per ottenere un help contestuale occorre associare alla la voce di menù o al componente un help ID utilizzando i metodi statici della CSH:

- **setHelpIDString** *(Component comp , String helpID)*
- **setHelpIDString** *(MenuItem menuItem, String helpID)*

Se vi sono diversi helpset è necessario specificare anche l'helpset utilizzando il metodo:

- **setHelpSet** *(Component comp, Helpset hs)*

che associa l'helpset hs al componente comp. Nel nostro esempio, *DemoHelp*, abbiamo inserito una toolbar con un pulsante. Si è poi provveduto a registrare come listener di tale pulsante un'istanza della classe *CSH.DisplayHelpAfterTracking*:

```
toolbarButton.addActionListener(new
                    CSH.DisplayHelpAfterTracking(hb);
```

Successivamente, si è impostato l'ID string relativa al componente (*textField*) presente sul pannello di *DemoHelp:*

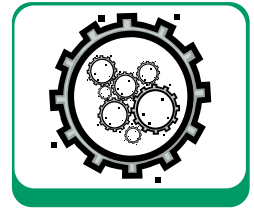
```
CSH.setHelpIDString(textField, "halloween");
```

Quando l'utente preme il pulsante presente sulla toolbar l'icona del mouse cambierà forma ad indicare che siamo nella modalità *HelpTracking*, a questo punto sarà sufficiente cliccare sul textfield per visualizzare la pagina di contenuto relativo all'ID halloween. Se il componente, di cui si richiede l'help, non ha associato alcun ID verrà mostrata la pagina iniziale dell'help.

CONCLUSIONI

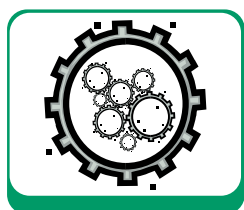
Questa volta mi sono contenuto, per evitare di annoiarvi. In questo articolo abbiamo illustrato come personalizzare ulteriormente il nostro help, come includere delle finestre secondarie ed effettuare il merge di più helpset. E' stato spiegato in che modo includere il nostro help all'interno di un'applicazione standalone e come si implementa l'help contestuale. Dopo la lettura di questo articolo resta un dubbio: come s'implementa il merge dinamico? Lascio a voi il compito di cimentarvi con questo problema… dandovi però un suggerimento: nella classe HelpSet ci sono dei metodi *add(HelpSet)* e *remove(HelpSet)* che potrebbero esservi utili. Spero che durante la lettura nessuno di voi si sia "perso"… e se così fosse vi invito a dare un'occhiata al programmino (molto essenziale) che troverete sul CD.
Alla prossima!

Giovanni Dodaro

CSH

Context Sensitive Help fornisce delle funzionalità per gestire l'help contestuale.

Controllare un ricevitore GPS

Un parser per il protocollo NMEA

La National Marine Electronics Association (NMEA) ha sviluppato uno standard che definisce la comunicazione tra una ricevente GPS ed un elaboratore.

Il sistema ha origini e finalità prevalentemente nautiche, è utilizzato per sistemi autopilota di imbarcazioni, per coprire le molteplici applicazioni marine e l'interfacciamento di un'ampia gamma di apparecchiature nautiche. I formati di dati del sistema NMEA sono numerosi, e solo una parte molto limitata ha rilevanza nell'ambito del GPS, dove il sistema NMEA viene prevalentemente impiegato per trasmettere dati da un ricevitore GPS verso un computer. L'idea del protocollo NMEA è quella di trasmettere in linea dei dati denominati "frasi", queste sono costruite da una sequenza di caratteri ASCII, indipendenti le une rispetto alle altre. NMEA non è l'unico protocollo di comunicazione tra Computers e GPS, effettivamente ne esistono diversi, basti pensare che quasi ogni costruttore di ricevitori ha il suo: Rockwell, SiRF, Garvin etc., molti di questi sono in formato binario, in modo da poter essere compressi e quindi occupare meno spazio. NMEA tuttavia è un protocollo molto efficiente e facilmente decodificabile grazie al suo formato.

Prefisso	Descrizione
$GPALM	Dati di Almanac
$GPAPA	Frase dell'autopilota A
$GP APB	Frase dell'autopilota B
$GPGGA	Dati relativi al Fix 3D
$GPGLL	Dati di Latitudine / Longitudine
$GPGSA	Dati satelliti generali
$GPGSV	Dati satelliti dettagliati
$GPRMB	Dati suggeriti di navigazione per i GPS
$GPRMC	Dati minimi suggeriti per il GPS
$GPRTE	Messaggio dell'itinerario
$GPVTG	Pista di vettore una velocità sopra la terra
$GPWCV	Velocità della chiusura di waypoint (velocità compensata)
$GPXTE	Errore trasversale misurato
$GPZDA	Data e tempo

TABELLA 1: Tipi di prefissi e relativa descrizione.

STRUTTURA DELLE FRASI NMEA

Studiare lo standard NMEA equivale a dire studiare le "frasi" dello standard NMEA, le analizzeremo in modo da conoscere meglio questo protocollo. Tutte le frasi dello standard hanno un prefisso, dato dalle prime due lettere che definiscono il tipo di dispositivo usato (per le riceventi GPS è *GP*), seguito da una sequenza di altre tre lettere che definiscono il tipo di frase. Ogni frase inoltre inizia con il carattere *'$'* e finisce con *CR LF* (*return/line*). La sequenza di stringhe può essere di massimo 80 caratteri (più i terminali della linea). I dati sono contenuti all'interno di questa singola linea con gli elementi separati da virgole, naturalmente questi possono variare nella quantità di precisione contenuta all'interno del messaggio, infine vi è una particolare sequenza di caratteri definita checksum che è preceduta dal simbolo '*'.

```
$PREFISSO,campo,campo,...,campo,*checksumCRLF
```

Il sistema è di tipo posizionale, i campi hanno una posizione ben precisa, infatti se i dati relativi ad un campo non sono disponibili, il campo può essere omesso, ma le virgole che lo delimiterebbero devono obbligatoriamente essere presenti, naturalmente tra le due virgole non deve esserci nessuno spazio.

I PREFISSI

Per capire bene i tipi di prefisso che può avere una frase NMEA basta analizzarne uno, ad esempio una frase che inizia con *$GPRMB*, indica che questa è inviata ad un dispositivo GPS (GP) ed è

di tipo RMB *(Recommended Minimum Navigation Info)*. Poiché ci stiamo occupando di GPS analizzeremo tutti i prefissi che iniziano per GP (ho escluso il carattere *'$'* che indica l'inizio della frase), nella Tab. 1 sono riportati alcuni tipi di prefissi con la relativa descrizione delle informazioni che possono contenere le loro frasi.

IL CHECKSUM

Per controllare eventuali errori durante la trasmissione dati, alla fine della frase dopo il carattere '*' è presente il checksum. Questo è un particolare metodo che assicura l'integrità della trasmissione. Il checksum è calcolabile facendo l'OR esclusivo a 8 bit di tutti i caratteri della frase NMEA, scartando quelli di inizio *($)* e di fine *(*)* ma includendo le virgole che servono a delimitare i campi. Da questo viene preso il valore esadecimale dei 4 bit più "alti" e più "bassi" e convertiti in due caratteri ASCII *(0-9, A-F)*. Vista la struttura generale delle frasi NMEA, le analizzeremo ora nel dettaglio, in particolare vedremo le frasi con prefisso $GPRMC e $GPGGA. Per avere ulteriori dettagli sul protocollo, la *National Marine Electronics Association*, che ha messo a punto lo standard, mette a disposizione la versione cartacea del protocollo, per informazioni potete visitare l'indirizzo internet *www.nmea.org* (lo stampato con le specifiche è acquistabile presso l'associazione ad un costo di circa *$300*)

```
$GPRMC - Recommended Minimum Specific
                          GPS/TRANSIT Data
```

RMC - NMEA è una delle frasi più complete del protocollo NMEA, questa ci da informazioni relative a data/ora, posizione e velocità, ci fornisce quindi tutti i dati necessari ad individuare un punto (*PVT: posizione,velocità, tempo)*. Esempio di una frase RMC:

```
$GPRMC,114753.007,A,3916.2905,N,01641.5780,E,
                  034.1,031.3,181103,002.1*6A
```

Nella Tab. 2 sono riassunti tutti i campi e di conseguenza tutte le informazioni che si possono acquisire con una frase di tipo GPRMC.

$GPGGA - Global Positioning System fix data

Analizziamo ora le frasi con prefisso *$GPGGA - Global Positioning System fix data. GGA – NMEA* comprende i dati relativi alla correzione differenziale ed al fix tridimensionale. Esempio di una frase GGA:

```
$GPGGA,114124.006,3916.3016,N,01641.5860,1,05,
                    04.2,01038.3,M,37.1,M,,,*66
```

FORMATO DEI CAMPI

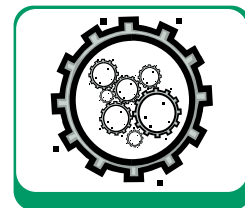
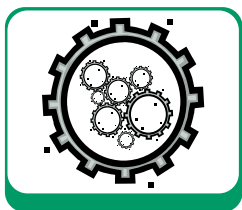
Sia nella Tab. 2 che nella Tab. 3 abbiamo visto diversi tipi di campi, conoscere il formato di questi è fondamentale. Iniziamo con l'ora, la misura del tempo è espressa in ore, minuti, secondi e centesimi di secondo (es: 114753.007= 11:47:53.007) relativi all'ora universale del sistema GMT, latitudine e longitudine sono espresse in gradi sessagesimali (es: 3916.3016 = 39°16.3016'), la velocità al suolo è espressa in Km/h, la data in giorno, mese ed anno (es: 181103 = 18 Novembre 2003), etc… I campi possono avere un diverso grado di precisione, questo dipende solo ed esclusivamente dalla capacità del ricevitore GPS, in quanto i dati inviati dal satellite sono gli stessi per tutti i ricevitori.

Campo	Descrizione	Commenti
$	Inizio frase	
GPRMC	Tipo di frase	
114753.007	Ora, riferita al sistema UTC	hhmmss.sss
A	Stato	A = Valid, V = Invalid
3916.2905	Latitudine della posizione attuale	ddmm.mmmm
N	Emisfero della posizione attuale	N=NORD, S=SUD
01641.5780	Longitudine della posizione attuale	ddmm.mmmm
E	Verso della posizione attuale	E=EST, W=OVEST
034.1	Velocità al suolo	nnn.nn
031.3	Corse	Nnn.nn Gradi reali
181103	Data	DDMMYY
003.1	Variazione / declinazione magnetica	Nnn.nn Gradi reali
*		
6°	Checksum	

TABELLA 2: I campi che si possono acquisire con una frase tipo GPRMC.

Campo	Descrizione	Commenti
$	Inizio frase	
GPGGA	Tipo di frase	
114753.007	Ora, riferita al sistema UTC	hhmmss.sss
3916.3016	Latitudine della posizione attuale	ddmm.mmmm
N	Emisfero della posizione attuale	N=NORD, S=SUD
01641.5860	Longitudine della posizione attuale	ddmm.mmmm
E	Verso della posizione attuale	E=EST, W=OVEST
1	Stato	0 = Invalid
		1 = GPS fix
		2 = DGPS fix
		3 = PPS fix
		4 = Real Time Kinematic
		5 = Float RTK
		6 = Estimated (dead reckoning)
		7 = Manual input mode
		8 = Simulation mode
05	Numero di satelliti in vista	
04.2	Horizontal Dilution Of Precision	
01038.3	Altitudine	
M	Unità di misura dell'altitudine	
37.1	Altezza del geoide rispetto all'ellissoide WGS84	
M	Unità di misura dell'altezza del geoide rispetto all'ellissoide	
	Ultimo aggiornamento DGPS	
	Identificatore della stazione DGPS	
66	Checksum	

TABELLA 3: Altri campi del GPRMC.

NMEA parser

Con le informazioni acquisite sopra, siamo ora in grado di costruire un parser per il protocollo NMEA, tramite l'implementazione di un applicativo in grado di estrarre le informazioni contenute nelle frasi dello standard. NMEA parser ha bisogno di un componente in grado di "dialogare" con la porta seriale (dove è collegato il ricevitore GPS), in modo da poter ricevere le frasi e successivamente decodificarle. Il controllo che fornisce all'applicazione funzioni per la comunicazione con la porta seriale è MSComm. MsComm va inizializzato: si imposta il numero della porta di comunicazione:

```
MSComm1.CommPort = 4
```

si impostano i parametri di velocità di trasmissione (4800), parità (Nessuna), bit di dati (8) e bit di arresto (1), in base a quanto stabilito dallo standard NMEA:

```
MSComm1.Settings = "4800,N,8,1"
```

si imposta il numero di caratteri che la proprietà Input legge dal buffer di ricezione, pari a zero in modo da far leggere al componente l'intero contenuto del buffer di ricezione:

```
MSComm1.InputLen = 0
```

ed infine si apre la porta di comunicazione:

```
MSComm1.PortOpen = True
```

Una volta configurato il controllo *MsComm*, l'applicativo è in grado di leggere dati dal ricevitore. I dati ricevuti sono una serie di caratteri ASCII visibile nel box laterale *"Esempio di sequenza"* da questa sequenza dovremmo estrarre due frasi una di tipo *GPRMC* e l'altra di tipo *GPGGA*, in modo da poter conoscere diversi parametri relativi alla nostra posizione (i parametri sono quelli delle tabelle 2 e 3, che descrivono le singole frasi).

Delimiter

Si dovrà creare una funzione *Delimiter* in grado di estrarre le singole frasi NMEA, dal buffer della porta seriale:

```
Private Function Delimiter(ByVal texte As String, ByVal
 delimiter1 As String, ByVal delimiter2 As String) As String
  Dim R1 As Integer
  Dim R2 As Integer
  Dim STT As String
  Dim STP As String
  Dim DDT As String
  DDT = texte
  STT = delimiter1
  STP = delimiter2
  R1 = CInt(InStr(1, DDT, STT))
  R2 = CInt(InStr(R1 + 1, DDT, STP))
  Delimiter = Mid(DDT, CLng(R1), CLng(R2 - R1))
End Function
```

Richiamando *Delimiter* come nel codice in basso

```
GPRMC_DATA = Delimiter(GPSDATA, "$GPRMC", vbCrLf)
GPGGA_DATA = Delimiter(GPSDATA, "$GPGGA", vbCrLf)
```

viene assegnato alle variabili *GPRMC_DATA* e *GPRMC_DATA* il contenuto delle frasi *GPRMC* e *GPGGA*.

PROCESS_RMC

Non ci resta che processare le frasi in modo da estrarre il contenuto dei loro campi. Questo compito è affidato alle funzioni *PROCESS_RMC* e *PROCESS_GGA*.

```
Private Function PROCESS_RMC(RMC As String)
                                        As Boolean
  Dim X As Integer
  Dim CHK As String
  For X = 1 To 12
    DoEvents
    RMC_DATA(X) = sGetToken(RMC, X)
    If X = 1 Then
      RMC_DATA(X) = Right(RMC_DATA(1), 5)
    End If
    If X = 12 Then
      RMC_DATA(X) = sGetToken(RMC, 2, "*")
    End If
  Next X
  CHK = GetChecksum(RMC)
  RMC_PREF = RMC_DATA(1)
  RMC_TIME = RMC_DATA(2)
  RMC_STATUS = RMC_DATA(3)
  RMC_LATITUDE = RMC_DATA(4)
  RMC_N_S = RMC_DATA(5)
  RMC_LONGITUDE = RMC_DATA(6)
  RMC_E_W = RMC_DATA(7)
  RMC_SPEED = RMC_DATA(8)
  RMC_COURSE = RMC_DATA(9)
  RMC_DATE = RMC_DATA(10)
  RMC_MAGNETIC = RMC_DATA(11)
  RMC_CHK_SUM = RMC_DATA(12)
  'Se checksum calcolata è ugauale a RMC_DATA(12)
      viene assegnato il valore TRUE alla funzione
  If RMC_CHK_SUM = CHK Then
    PROCESS_RMC = True
  Else
    PROCESS_RMC = False
  End If
End Function
```

ESEMPIO DI SEQUENZA

$GPGGA,145903.000,3916.3036,N,01641.5976,E,0,00,,,M,,M,,*4DCRLF$GPGSA,A,1,,,,,,,,,,,,,,*1ECRLF$GPGSV,1,1,04,25,30,298,,05,39,106,,14,08,240,,02,20,296,*74CRLF............$GPRMC,145903.000,V,3916.3036,N,01641.5976,E,,,191103,,*11CRLF$GPGSV,1,1,04,14,08,240,,24,19,051,,30,75,091,,18,09,182,*7ECRLF$GPRMC,145946.000,V,3916.3036,N,01641.5976,E,,,191103,,*10;

La funzione restituisce un valore booleano in base al checksum della frase NMEA in modo da conoscere, dal valore restituito da questa funzione, se ci sono stati errori nella ricezione dei dati. Le informazioni fornite dal ricevitore GPS attraverso i campi della frase GPRMC sono contenuti in un Array RMG_DATA di dodici elementi, uno per ogni campo della frase. La Tab. 2 ci fornisce la documentazione necessaria al riconoscimento dei campi. Basta quindi assegnare alle variabili globali *RMC_PREF, RMC_TIME, RMC_STATUS, RMC_LATITUDE, RMC_N_S, RMC_LONGITUDE, RMC_E_W, RMC_SPEED, RMC_COURSE, RMC_DATE, RMC_MAGNETIC e RMC_CHK_ SUM*, rispettivamente il corrispondente valore contenuto nel vettore per conoscere i dati fornito dal ricevitore satellitare. La funzione *sGetToken* che ha il compito di estrarre dalla frase il singolo *Token* (il codice completo lo trovate sul Web)*:*

```
Function sGetToken(ByVal sAllTokens As String,
    Optional ByVal iToken As Integer = 1, Optional ByVal
                            sDelim As String = ",") As String
    Static iCurTokenLocation As Long
    Static nDelim As Integer
    nDelim = Len(sDelim)
    If iToken < 1 Or nDelim < 1 Then
        Exit Function
        …
    End If
End Function
```

PROCESS_GGA

La funzione PROCESS_GGA, processa le frasi di tipo GPGGA:

```
Function PROCESS_GGA(GGA As String) As Boolean
    Dim X As Integer
    Dim CHK As String
    For X = 1 To 16
        DoEvents
        GGA_DATA(X) = sGetToken(GGA, X)
        If X = 16 Then
            GGA_DATA(X) = sGetToken(GGA, 2, "*")
        End If
        If X = 1 Then
            GGA_DATA(X) = Right(GGA_DATA(1), 5)
        End If
    Next X
    CHK = GetChecksum(GGA)
    GGA_STATUS = GGA_DATA(7)
    GPS_NUM_SAT = GGA_DATA(8)
    GPS_ALTITUDE = GGA_DATA(10)
    GGA_CHK_SUM = GGA_DATA(16)
    If GGA_CHK_SUM = CHK Then
        PROCESS_GGA = True
    Else
        PROCESS_GGA = False
```

```
    End If
End Function
```

Checksum

Infine, non ci resta che analizzare come viene controllato il checksum delle frasi:

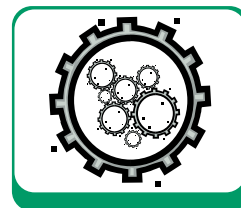
```
Function GetChecksum(ByRef sInString As String) As String
    Dim lCurrent&, lLast&
    On Error Resume Next
    If Mid$(sInString, 1, 1) = "$" Then
        sInString = Mid$(sInString, 2)
    End If
    lLast& = Asc(Mid$(sInString, 1, 1))
    For lCurrent& = 2 To Len(sInString) - 3
        lLast& = lLast& Xor Asc(Mid$(sInString,
lCurrent&, 1))
    Next
    GetChecksum = CStr(Hex(lLast&))
End Function
```

Le variabili globali *GGA_CHK_SUM* e *RMC_CHK_SUM* contengono il valore del checksum acquisito dal ricevitore GPS, per conoscere quindi se ci sono stati errori durante la comunicazione, basta confrontare questi valori con quelli restituiti dalla funzione *GetChecksum* passandoci come riferimento la frase NMEA.

CONCLUSIONI

L'intero articolo sul protocollo NMEA e sull'implementazione di un suo Parser è stato scritto, oltre che per conoscere più da vicino il protocollo di comunicazione tra Computers e ricevitori GPS, anche per poter interfacciare il proprio ricevitore con un proprio software, permettendoci di eliminare tutti gli activeX e le Dll prodotti da altri (e limitati!!!).

Nel precedente numero di ioProgrammo abbiamo parlato di GPS mediante l'utilizzo di un ricevitore satellitare collegato ad un Notebook, in particolare abbiamo monitorato la nostra posizione geografica nel tempo, utilizzando per l'interfaccia tra ricevitore e Computers il componente *GpsToolsXp* della Franson, in versione shareware limitata all'utilizzo di soli 14 giorni. Ora non abbiamo più bisogno di questo poiché attraverso la dll (NmeaParser.dll) che si può creare mettendo insieme le funzioni viste sopra, è possibile ricevere e decodificare attraverso il nostro computers tutte le informazioni rilevabili da un ricevitore GPS. Sul sito Web di ioProgrammo (*cd-rom. ioprogrammo.it*) troverete la nuova versione di *GpsNavigator,* ma questa volta senza nessuna limitazione!!!

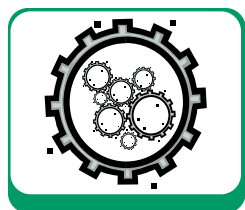
Luigi Salerno

ELENCO COMPLETO DELLE VARIABILI GLOBALI DI PROCESS-GCA

Come per la funzione PROCESS_RMC anche PROCESS_GGA ha una serie di variabili globali: GGA_ PREF, GGA_TIME, GGA_LATITUDE, ecc. che contengono altri dati forniti dal ricevitore GPS. È necessario quindi creare una funzione per ogni tipo di frase, poiché queste variano sia nel numero di campi (una frase di tipo RMC ha 12 campi, una GGA ne ha 16), che nel tipo.

Zip: creazione ed estrazione di file

Un WinZip con Java

parte seconda

Iniziamo lo sviluppo di un programma Java con interfaccia grafica per molti versi simile al noto WinZip. L'utente potrà gestire tutte le principali caratteristiche di un archivio ZIP con pochi colpi di mouse.

Nel corso della prima parte di questo tutorial abbiamo conosciuto i contenuti del package *java.util.zip*, una tra le tante collezioni di API che fanno parte della piattaforma *Java 2 Standard Edition* (J2SE). Per dimostrare le funzionalità e per approcciare i meccanismi d'uso di questo set di API sono state realizzate due applicazioni Java per riga di comando: una in grado di estrarre qualsiasi archivio ZIP esistente, l'altra capace di crearne uno ex novo. Questa seconda parte del tutorial si concentra sullo sviluppo di una *GraphicalUser Interface* (GUI) per la gestione degli archivi compressi ZIP. Per dirla con parole più semplici, imiteremo un programma come il noto WinZip, stendendo le basi per la realizzazione di un gestore di archivi compressi con interfaccia grafica. Giacché stiamo lavorando con Java, otterremo un software assolutamente multipiattaforma, che potrà essere usato sopra qualsiasi moderno sistema operativo.

SWINGZIP

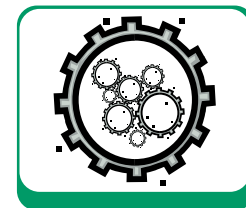
Per lo sviluppo dell'interfaccia grafica impiegheremo i componenti Swing di Java. Da qui il nome dell'applicazione: *SwingZIP*. Prima di iniziare a scrivere del codice, formalizziamo il nostro progetto. Essendo la prima applicazione di questo genere che andiamo a realizzare, non è nostra intenzione strafare. Ci accontenteremo di poche ma efficenti funzionalità, e non cureremo troppo l'aspetto e la completezza dell'interfaccia grafica. D'altra parte, successivamente avremo tutto il tempo che vorremo per estendere, completare e rifinire l'applicazione. Stabiliamo alcuni punti chiave:

- L'applicazione sarà contenuta in un'unica finestra. Nella parte alta avremo una barra degli strumenti in cui inserire i pulsanti di azione. Nella parte centrale, avremo un componente capace di mostrare il contenuto dell'archivio di volta in volta gestito. Nella parte bassa inseriremo una barra di progresso per tenere informato l'utente sullo stato delle operazioni.
- Le azioni consentite saranno: creare un nuovo archivio, aprire un archivio esistente, estrarre uno o più file dall'archivio corrente, aggiungere uno o più file all'archivio corrente e, infine, rimuovere uno o più file dall'archivio corrente. Cinque operazioni fondamentali che consentono comunque una completa gestione di un archivio compresso ZIP.

Bene, giunti a questo punto, forti delle nozioni acquisite con la prima parte del tutorial, possiamo iniziare a scrivere del codice.

LO SCHELETRO DELL'APPLICAZIONE

Cominciamo assemblando un'interfaccia grafica capace di soddisfare i requisiti stabiliti nel corso del paragrafo precedente. Non ci vuole poi molto, il codice completo lo trovate in (*\passo1\swingzip .java)*:

```
// Per l'interfaccia grafica.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
// Per manipolare file e archivi ZIP.
import java.io.*;
import java.util.zip.*;
// Utilità.
import java.util.*;
// La classe fondamentale dell'applicazione.
```

```
public class SwingZIP extends JFrame implements
                                        ActionListener {
  // I pulsanti della toolbar.
  JButton b1 = new JButton("Crea");
  JButton b2 = new JButton("Apri");
  JButton b3 = new JButton("Estrai");
  JButton b4 = new JButton("Aggiungi");
  JButton b5 = new JButton("Elimina");
  // La lista dei file contenuti nell'archivio.
  JList filesList = new JList();
  // La barra di stato per il progresso delle operazioni.
  JProgressBar progressBar = new JProgressBar(0, 100);
  // Un riferimento verso il file gestito.
  File file = null;
  // Costruttore.
  public SwingZIP() {
    super("SwingZIP");
    // Imposta le proprietà dei componenti.
    …
    // Aggiunge il pannello alla finestra.
    getContentPane().add(all);
    // Compatta la finestra.
    pack();
    // Posizione al centro dello schermo.
    Dimension screenSize =
          Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = getSize();
    setLocation((screenSize.width - frameSize.width)
      / 2, (screenSize.height - frameSize.height) / 2);
    // Quando l'utente chiude la finestra, termina il
                                            programma.
    setDefaultCloseOperation(EXIT_ON_CLOSE); }
  // Metodo richiamato alla pressione di uno dei pulsanti.
  public void actionPerformed(ActionEvent e) {
    // Individua il pulsante premuto.
    JButton b = (JButton)e.getSource();
    // Intraprende un'azione differente a seconda del
                                      pulsante premuto.
    if (b == b1) zipCreate();   // Crea nuovo archivio.
    else if (b == b2) zipOpen();    // Apri archivio
                                            esistente.
    else if (b == b3) zipExtract(); // Estrai file
                                dall'archivio corrente.
    else if (b == b4) zipAdd();     // Aggiungi file
                                 all'archivio corrente.
    else if (b == b5) zipDelete();  // Elimina file
                              dall'archivio corrente. }
  // Crea un nuovo archivio.
  private void zipCreate() {}
  // Apre un archivio esistente.
  private void zipOpen() {}
  // Estrae file dall'archivio corrente.
  private void zipExtract() {}
  // Aggiunge file all'archivio corrente.
  private void zipAdd() {}
  // Elimina file dall'archivio corrente.
  private void zipDelete() {}
  // Metodo di utilità interna usato per mostrare
                                   messaggi all'utente.
  private void showMessage(String title, String
                               message, int type) {
    JOptionPane.showMessageDialog(this, message,
                                       title, type); }
  // Punto di avvio dell'applicazione.
  public static void main(String[] args) {
    // Crea la finestra e la visualizza.
    SwingZIP swingzip = new SwingZIP();
    swingzip.show(); }}
```

La classe *SwingZIP*, in questa forma, può già essere compilata ed eseguita. Se ne ricaverà qualcosa di estremamente simile a quanto mostrato in Fig. 1. Come si osserva, l'interfaccia dell'applicazione è spartana ma efficace. Nella parte alta della finestra è stata introdotta una barra degli strumenti che propone un pulsante per ciascuna delle operazioni fondamentali stabilite in precedenza. Nella parte centrale fa mostra di sé un oggetto *JList*, il cui compito sarà mostrare l'elenco dei file contenuti nell'archivio che di volta in volta sarà gestito dall'applicazione. Infine, nella parte più bassa delle finestra, c'è una barra di progresso: sarà utile per far capire all'utente quanto ci vorrà per completare le operazioni più dispendiose di risorse temporali. Benché, a giudicare dall'interfaccia, il programma appaia già completo, attivando uno dei due tasti abilitati all'uso (*"Crea"* e *"Apri"*) non accadrà nulla. Scorrendo il codice mostrato sopra, localizzate il metodo *actionPerformed()*, che è quello che viene richiamato alla pressione di uno dei bottoni, ciascuno dei cinque pulsanti disponibili è stato associato ad un metodo della classe, il corpo dei metodi invocati è vuoto. Questi cinque metodi privati costituiscono il "motore" del programma, cioè la parte di cui ci andremo ad occupare da questo momento in poi, completandola piano piano.

APERTURA DI UN ARCHIVIO

Cominciamo dall'apertura di un archivio esistente, che corrisponde al metodo *zipOpen()*. La prima cosa che il codice di questa funzione dovrà effettuare sarà far scegliere all'utente quale archivio aprire. Per far ciò possiamo sfruttare la classe *JFileChooser* di Swing, capace di creare e visualizzare automaticamente una finestra di dialogo per aprire o salvare un file. Prima di servircene, però, andiamo a realizzare una classe esterna, chiamata *ZIPFileFilter*:

```
import java.io.File;
import javax.swing.filechooser.FileFilter;
class ZIPFileFilter extends FileFilter {
  // Richiamato per decretare se il file in argomento
                                  debba essere mostrato.
```

```
public boolean accept(File f) {
    if (f.isDirectory()) return true;
    String fileName = f.getName().toLowerCase();
    return fileName.endsWith(".zip"); }
// Richiamato per ottenere una descrizione del filtro.
public String getDescription() {
    return "Archivi ZIP (*.zip)"; } }
```

ZIPFileFilter estende la classe *javax.swing.filechooser.FileFilter*. Lo scopo di un *FileFilter* è aiutare un *JFileChooser* nel decidere quali file visualizzare e quali nascondere. Associando un oggetto *ZIPFileFilter* al *JFileChooser* che andremo a breve ad impiegare, faremo in modo che l'utente visualizzi solo le directory ed i file con estensione *.zip*. Fatto ciò, posizioniamoci all'interno del metodo *zipOpen()* di *SwingZIP* e cominciamo ad allestirne il codice:

Fig. 1: L'interfaccia grafica di SwingZIP. Semplice ma funzionale!

```
// Fa scegliere il file all'utente.
JFileChooser fileChooser = new JFileChooser();
fileChooser.setFileFilter(new ZIPFileFilter());
int option = fileChooser.showOpenDialog(this);
// Controlla che l'utente non abbia annullato l'operazione.
if (option != JFileChooser.APPROVE_OPTION) return;
// Acquisisce il file selezionato.
final File selectedFile = fileChooser.getSelectedFile();
```

Ricompilate il sorgente ed avviate l'applicazione. Alla pressione del pulsante "*Apri*" apparirà ora una finestra di dialogo simile a quella mostrata in Fig. 2. Un elemento del codice appena mostrato può aver incuriosito i lettori più attenti: perché la variabile *selectedFile*, il cui compito è memorizzare il riferimento verso il file selezionato dall'utente, è stata definita *final?* Abbiate pazienza, sarà spiegato tra un altro po'. Ammettiamo che l'archivio ZIP scelto dal nostro utente sia piuttosto grande e comprenda numerose *entry*. Sicuramente, in un caso come questo, l'applicazione impiegherà qualche secondo per esaminarlo tutto e per visualizzare la lista dei suoi contenuti. Pertanto è bene fare in modo che l'analisi sia eseguita all'interno di un thread separato, per evitare che l'interfaccia grafica rimanga impegnata più a lungo del dovuto. In caso contrario, l'utente vedrebbe il programma congelarsi, poiché la finestra non sarebbe più in grado di rispondere alle sue sollecitazioni. L'utente potrebbe addirittura credere che il software si sia bloccato. Seguite sempre questa regola aurea: quando un'operazione comandata da interfaccia grafica può richiedere svariati secondi per il suo completamento, lanciate sempre un thread secondario e assicuratevi che l'operazione venga eseguita all'interno di quest'ultimo. Il software sembrerà più efficiente. Prima di lanciare il nostro thread secondario per l'analisi del file prescelto, compiamo alcune operazioni preliminari:

Fig. 2: La finestra di dialogo per l'apertura di un archivio ZIP.

```
// Si rende inattiva l'interfaccia grafica.
...
b5.setEnabled(false);
// Si perde il riferimento all'archivio precedente.
file = null;
// Si svuota la lista attualmente mostrata.
filesList.setListData(new Object[0]);
// Si resetta il titolo della finestra.
setTitle("SwingZIP");
```

Per prima cosa, vengono disabilitati tutti e cinque i pulsanti del software. In questo modo, l'utente non potrà comandare ulteriori operazioni finché il thread che a breve lanceremo resterà in esecuzione. Se così non fosse, potrebbe capitare che l'utente tenti di aprire due file simultaneamente! Consideriamo poi la possibilità che il metodo *zipOpen()* sia richiamato non di seguito all'avvio del programma, ma dopo che un altro archivio sia già stato aperto e gestito. Bisogna ripulire l'oggetto *SwingZIP* corrente da tutti i riferimenti verso un archivio precedentemente aperto, prima di operare su uno nuovo. Pertanto, la proprietà *file* (che serve per mantenere un riferimento verso l'archivio corrente) viene annullata, l'elenco dei contenuti svuotato ed il titolo della finestra reimpostato (faremo in modo che, dopo l'apertura di un archivio, il nome del file corrispondente appaia nel titolo della finestra).
Dobbiamo ora imbastire il thread secondario da dedicare esclusivamente alle operazioni di apertura dell'archivio scelto. Per non dover scrivere altre classi, e per poter rimanere a lavorare all'interno di *zipOpen()*, sfrutteremo una inner-class anonima. Ecco il modello da applicare al caso specifico:

```
// Crea un thread per gestire al suo interno l'apertura.
Thread thread = new Thread(new Runnable() {
    public void run() {
        // Codice per aprire l'archivio.}
});
// Si avvia il nuovo thread.
thread.start();
```

Se avete pratica con l'utilizzo delle inner-class anonime allora saprete già che una classe di questo tipo

può usare le variabili locali del metodo che la contiene solo se queste sono di tipo *final*. Ecco perché *selectedFile* è stata resa immutabile: ci servirà all'interno del thread secondario per capire quale file debba essere aperto. Proseguiamo popolando il metodo *run()* del nuovo thread con il codice capace di realizzare l'apertura e l'analisi dell'archivio:

```
public void run() {
 // Dichiara l'oggetto ZipFile che sarà istanziato a breve.
 ZipFile zipFile = null;
 // Inizia la sezione a rischio di eccezione.
 try {
   // Crea l'oggetto ZipFile per la decodifica del file
                                                   scelto.
   zipFile = new ZipFile(selectedFile);
   // Quante entry ci sono nell'archivio?
   int size = zipFile.size();
   // Dichiara l'array delle entry.
   ZipEntry[] entries = new ZipEntry[size];
   // Recupera l'enumerazione delle entry.
   Enumeration entriesEnumeration = zipFile.entries();
   // Passa in rassegna le entry e le copia nell'array.
   for (int i = 0; i < size; i++) {
     // Acquisisce l'entry.
     entries[i] = (ZipEntry)entriesEnumeration
                                            .nextElement();
     // Aggiorna la barra di progresso.
     progressBar.setValue((int)Math.round((
                              (i + 1) * 100D) / size)); }
   // L'archivio è stato aperto.
   // Si associa la proprietà file.
   file = selectedFile;
   // Si mostra l'elenco dei contenuti.
   filesList.setListData(entries);
   // Si mette il nome dell'archivio nel titolo della finestra.
   setTitle(file.getName() + " - SwingZIP");
   // Si abilitano i pulsanti "Estrai", "Aggiungi"
                                             e "Elimina".
   b3.setEnabled(true);
   b4.setEnabled(true);
   b5.setEnabled(true);
 } catch (IOException e) {
 // Non si riesce a leggere il file come archivio ZIP.
 showMessage("Errore", "Impossibile leggere il file
       selezionato",JOptionPane.ERROR_MESSAGE);
 } finally {
   // Chiude lo ZipFile, se è stato aperto.
   if (zipFile != null) try {
     zipFile.close();
   } catch (Exception e) {}
   // Riporta sullo zero la barra di progresso.
   progressBar.setValue(0);
   // Si riattivamo i pulsanti "Crea" e "Apri".
   b1.setEnabled(true);
   b2.setEnabled(true); } }
```

Nel corso del primo appuntamento di questo tutorial abbiamo già imparato come utilizzare le API del package *java.util.zip*, pertanto il codice appena mostrato contiene poco di nuovo. Tutto si muove al seguente modo:

- Viene creato un oggetto ZipFile, per la decodifica dell'archivio. Se la creazione fallisce si propaga una *IOException* (o una *ZipException*, che comunque deriva da *IOException*). In questo caso il file non può essere letto, ed è necessario avvisare l'utente (blocco *catch*).
- Se l'apertura del file riesce senza problemi, bisogna passare in rassegna le entry dell'archivio. Le entry vengono valutate e trasferite in un array. Man mano che l'analisi viene eseguita, la barra di progresso, sulla parte bassa della finestra, viene aggiornata per riflettere lo stato corrente dell'operazione. L'utente, in questo modo, capirà che il programma sta lavorando per lui, ed inoltre potrà stimare la durata dell'attesa.
- Al termine dell'analisi, il file aperto viene registrato come archivio corrente, il suo contenuto viene mostrato dall'oggetto *JList* preposto ed il suo nome entra a far parte del titolo della finestra. I pulsanti *"Estrai"*, *"Aggiungi"* ed *"Elimina"* vengono attivati. Ora che c'è un archivio aperto l'utente ha tutto il diritto di usarli!
- Comunque siano andati i passi precedenti, al termine dell'operazione la barra di progresso viene rimessa sullo zero, mentre i pulsanti *"Crea"* ed *"Apri"* vengono riabilitati (blocco *finally*).

Abbiamo fatto! Il metodo *zipOpen()* è ora completo. Sperimentate pure l'apertura di un archivio *ZIP*. Dovreste ottenere un risultato grafico simile a quello di Fig. 3.

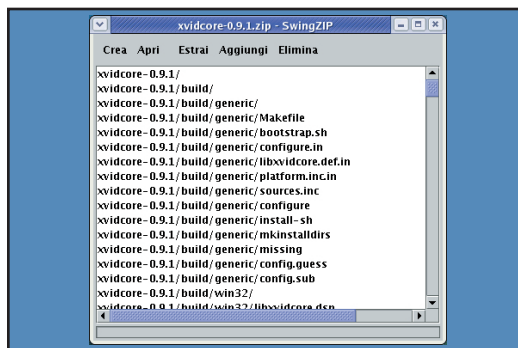
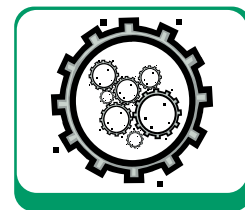



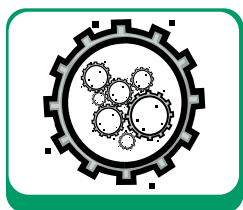
Fig. 3: ***SwingZIPmostra i contenuti di un archivio ZIP.***

IL MESE PROSSIMO...

Il mese prossimo proseguiremo da dove abbiamo appena interrotto. Ora che l'operazione di apertura di un archivio è completa, vedremo come fare per estrarre i file contenuti al suo interno. Vi aspetto!

Carlo Pelliccia

Le basi della programmazione multimediale su .NET

Introduzione alle DirectX su .NET

Al termine di questo articolo si sarà in grado di inizializzare DirectX in .NET e si conosceranno i concetti base per lo sviluppo di applicazioni multimediali su Windows.

La conoscenza della sintassi C# e delle tecniche di programmazione ad oggetti è essenziale. Esperienza nel compilare ed eseguire progetti con Microsoft Visual Studio .NET è d'obbligo. Una conoscenza di base di matematica è anche consigliabile.

Nella foto qui sotto sono mostrati alcuni esempi di applicazioni pratiche di DirectX: videogiochi, software per la sanità, realtà virtuale, animazioni, software di disegno, ecc. È opportuno assimilare bene i concetti base che tratteremo in questo articolo e che costituiranno i mattoni fondamentali dello sviluppo di applicazioni .NET multimediali.

Fig. 1: **Alcuni esempi di applicazioni con DirectX.**

DIRECTX: COSA SONO?

Microsoft DirectX è una avanzata suite di API per applicazioni multimediali che gira sui sistemi operativi Microsoft Windows. DirectX è una piattaforma di sviluppo standard che consente agli sviluppatori software di accedere a specifiche caratteristiche hardware senza bisogno di scrivere codice specifico per l'hardware. Questa tecnologia fu introdotta nel 1995 (con il nome di *GAME SDK*) ed è riconosciuta come standard per lo sviluppo di applicazioni multimediali su piattaforma Windows. DirectX offre il miglior risultato possibile con la grafica, il suono, la musica e le animazioni e consente al programmatore di accedere a caratteristiche hardware, come l'accelerazione 3D e avanzati sistemi di audio, mantenendo sempre la stessa interfaccia. Quindi attraverso questa libreria, si prescinde dalla programmazione specifica delle caratteristiche dei diversi hardware presenti sui computer, sfruttando però sempre al meglio le caratteristiche specifiche di ogni hardware. Su *www.microsoft.com/directx* è possibile scaricare l'ultima versione di DirectX (la 9.0 al momento) e, se Microsoft tornerà finalmente a concederci la licenza, potrete trovarle in uno dei prossimi numeri di ioProgrammo. Questo download include le API DirectX, l'interfaccia *Managed DirectX* necessaria ai linguaggi .NET per poterci lavorare, l'SDK *(Software Development Kit)* e un esaustivo insieme di esempi e documentazione. Quando si programma DirectX 9.0 o versioni precedenti *non-managed* (ossia pre-.NET), bisogna prima di tutto creare un oggetto principale di tipo *DirectXn*, dove n sta per il numero della versione (ad esempio DirectX8 per la versione 8.1), e da questo si possono poi creare tutti gli altri oggetti secondari specifici che richiederà la nostra applicazione. Nella versione *Managed* di DirectX, possiamo invece direttamente creare gli oggetti secondari, che sono:

Direct3D: per accedere alle funzioni di accelerazione 3D.
Direct3DX: per accedere a funzioni per facilitare la scrittura del codice (come funzioni per la moltiplicazione di matrici).
DirectDraw: inserita per compatibilità con i vecchi programmi, dato che non ci sono nuove caratteristiche in questa versione. È essenzialmente utilizzata per creare giochi 2D.
DirectInput: per controllare tutti i device di Input, da mouse e tastiera a joystick con *"force feedback"* (ossia interattivi) e strumenti di realtà virtuale come casco e guanti.

DirectPlay: per la creazione di giochi multiplayer via network.
DirectAudio: per la manipolazione ed esecuzione di tutti i tipi di suoni, che comprende *DirectSound* per l'esecuzione di effetti sonori (file *wav*) e *DirectMusic* per le colonne sonore (file *midi*). Attualmente, nella versione 9.0 di DirectX, solo *DirectSound* ha un'interfaccia *managed*.
DirectShow: per l'esecuzione e la cattura di video e audio in formati particolari (come *mp3*).
DirectSetup: che consiste nelle API di DirectX per la distribuzione dei pacchetti di installazione dei software basati su DirectX.

Nei primi articoli tratteremo Direct3D.

IL MOTORE 3D DELLE DIRECTX: DIRECT3D

Quando programmiamo applicazioni grafiche 3D, stiamo essenzialmente lavorando con due distinti tipi di risorsa: dati geometrici e colori, o classificandoli secondo la loro rappresentazione: vertici e pixel. Il software che esegue questo lavoro è chiamato Motore 3D *(3D Engine)*, e in questo articolo capiremo come DirectX giochi un ruolo essenziale in questo motore. Consideriamo una semplice figura geometrica, il cubo. Rispettando il piano cartesiano un cubo può essere costruito con 8 punti, che chiamiamo vertici. La struttura di un vertice può essere rappresentata nel seguente modo:

- **x, y, z:** punti del sistema cartesiano;
- **c:** il colore del vertice;
- **u, v:** cordinate della texture di questo vertice, che vedremo nei prossimi articoli.

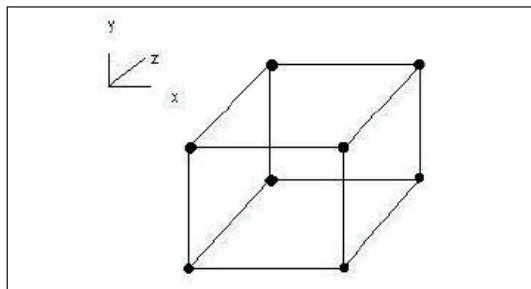



***Fig. 2:** Gli 8 vertici del cubo e le rispettive 6 facce quadrate.*

Connettendo i vertici fra loro, vediamo chiaramente che ci sono 6 distinte facce, che assieme formano la figura del cubo. Inoltre, notiamo che ogni faccia è un poligono piano ed ha la figura di un rettangolo. Se volessimo, potremmo dividere ciascuno di questi rettangoli connettendo due dei suoi vertici opposti per produrre due triangoli adiacenti, senza bisogno di dover fornire nuove coordinate per altri vertici. Notiamo che mentre i triangoli sono sempre rappresentati dagli 8 vertici dai quali eravamo partiti, c'è stato un cambiamento nella connessione fra questi vertici. Questa differenza è molto importante, perché Direct3D usa i triangoli per disegnare qualsiasi cosa. Questo sia perché il triangolo è la figura più semplice e quindi veloce da rappresentare, e sia perché è sicuro che un triangolo si possa trovare esclusivamente in un solo piano. A questo punto, possiamo tranquillamente capire il concetto di *Mesh*. Per mesh si intende l'insieme dei vertici e delle informazioni di connessione necessarie per produrre un oggetto tridimensionale (come il nostro cubo). Una o più mesh disposte in uno stesso mondo virtuale, insieme alle luci ed altri elementi, formano una Scena. In fine introduciamo il concetto di *Frame*. Se andiamo al cinema, vediamo che la pellicola è formata da un susseguirsi di tante immagini che, viste una dopo l'altra ad elevata velocità, danno l'illusione di non esser più un susseguirsi di immagini statiche, ma di un'unica immagine in movimento. Quello che fa un motore 3D è essenzialmente la stessa cosa: usa una telecamera virtuale per catturare serie cronologiche di immagini digitali di un mondo virtuale 3D. Queste immagini vengono poi presentate in un display 2D, il monitor del computer, ad una velocità tale da creare la sensazione di movimento e animazione. Ad un livello hardware, esistono le schede grafiche 3D con i loro *display device*. Ogni scheda contiene un oggetto fisico di hardware chiamato *Adapter*, che da supporto per vari display mode (risoluzione dello schermo, profondità di bit). In alcuni casi sulle schede grafiche sono disponibili più di un adapter. Le schede grafiche hanno inoltre della memoria fisica per l'immagazzinamento di dati. La porzione di questa memoria che verrà usata per l'immagazzinamento dei dati che saranno presentati sullo schermo è chiamata *Frame Buffer*. Essenzialmente questo è un array di valori di colore *(pixel)* che saranno convertiti negli appropriati segnali elettrici per la riproduzione dell'immagine sullo schermo. Come è possibile immaginare, ad alto livello, il lavoro di applicazione 3D sarà di riempire, attraverso DirectX, questo array di memoria con dei dati appropriati. Per ottenere i vantaggi dell'accelerazione 3D hardware delle moderne schede grafiche, bisogna chiaramente sapere come comunicare con l'hardware dalla propria applicazione. Come noto, i driver sono il software che fornisce

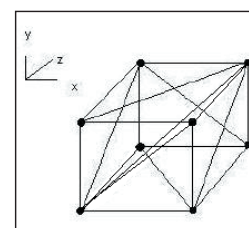
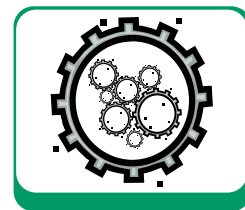



***Fig. 3:** Gli 8 vertici del cubo e le rispettive 12 facce triangolari.*

NOTA

DIRECTDRAW

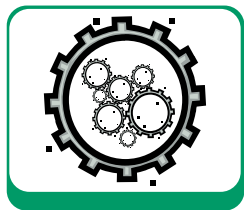
**DirectDraw è una libreria molto potente e non va immaginata superata solo perché si occupa del 2D: ad esempio con una sola semplicissima funzione è possibile girare di 180 gradi la scena che si sta rappresentando sullo schermo.
Se qualcuno ricorda il primo "Prince Of Persia" può immaginare quanto avrebbe aiutato gli sviluppatori del gioco quando il protagonista, dopo aver bevuto la pozione magica, vede tutto sottosopra. Gli sarebbe bastato richiamare un API e continuare a gestire tutto allo stesso modo. I programmatori dovettero invece scriversi delle loro funzioni che, frame per frame, consideravano le collisioni al contrario, una funzione per il calcolo della gravità inversa, ecc.**

l'interfaccia ed espone le funzionalità disponibili nell'hardware. Dato che praticamente ogni scheda grafica usa il proprio set di driver, sarebbe davvero proibitivo gestire un'applicazione che dialoghi direttamente con i driver di ogni scheda grafica in commercio. Ed è qui che DirectX diventa così importante, dato che espone sempre la stessa interfaccia senza doversi quindi preoccupare di quale strato di hardware ci sia sotto. Come si vede da Fig. 4 DirectX rappresenta lo strato fra l'applicazione e l'hardware grafico (attraverso i suoi driver). Questo permette, quando si sviluppa, di concentrarsi su i compiti che dovrà svolgere l'applicazione, invece di preoccuparsi di supportare la piattaforma sottostante.

Fig. 4: L'architettura di Direct3D.

INIZIALIZZAZIONE

Classi .NET

La prima cosa comoda e utile da fare, è lavorare nel namespace di DirectX e DirectX.Direct3D:

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
```

Adapter

E' possibile elencare ed ottenere informazioni su tutti gli Adapter presenti sulla macchina. Per un dettagliato esempio su come enumerare gli adapter fate riferimento all'esempio di Microsoft che, una volta installato DirectX 9.0, troverete su *"(SDK-root)\Samples\C#\Common\D3DEnumeration.cs"*. Per ogni adapter l'applicazione enumera i display mode che supporta attraverso la proprietà Adapters dell'oggetto Manager. La collezione di adapters può essere esaminata attraverso l'oggetto *AdapterListEnumerator.* Noi faremo riferimento soltanto all'*Adapter di default,* per non complicare eccessivamente il codice, considerato anche che è quasi sempre la scelta ideale e più comune.

Device

Il *Device* è un oggetto speciale di DirectX, che ci permette di accedere alle caratteristiche di accelerazione 3D. I Device disponibili sono:

Hardware *(Hardware Abstraction Layer):* Con questo device abbiamo diretto accesso all'accelerazione hardware, ed è la scelta ottimale per le prestazioni. Se però proviamo a creare questo device ma non abbiamo una scheda con accelerazione 3D DirectX andrà in errore e lancerà un eccezione.
Reference *(Reference Rasterizer):* Questo device è incluso nell'SDK di DirectX e non dipende dal supporto hardware. È molto utile per testare e provare il software, ma allo stesso tempo è estremamente lento. Deve essere usato solo per scopi di debug.
Software *(Software Device):* Usato solo in mancanza di schede con accelerazione 3D.

Quando si crea un device bisogna specificare il tipo di Adapter, l'handle della finestra che sarà usato da DirectX, e altri due parametri che specificano i dettagli del device, ossia i flag di comportamento e i parametri di presentazione.

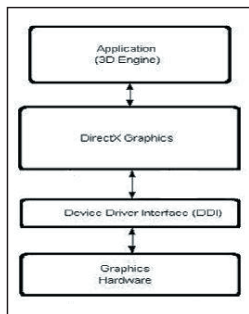
```
Device device = null;
device = new Device(Manager.Adapters.Default.Adapter,
                                    DeviceType.Hardware,this,
    CreateFlags.SoftwareVertexProcessing, presentParams);
```

I flag di comportamento principali sono:

SoftwareVertexProcessing: Questa opzione è sempre disponibile, ma è la più lenta. Comunica a DirectX che tutti i calcoli necessari avverranno via software, ignorando l'accelerazione 3D hardware.
HardwareVertexProcessing: Con questo flag si forza DirectX ad usare l'accelerazione hardware per il calcolo dei vertici, ma lasciando altre operazioni come lo shading e l'illuminazione al livello software. Se non è presente una piattaforma hardware con accelerazione 3D la creazione del device fallirà.
MixedVertexProcessing: Come suggerisce il nome stesso, con questa opzione si useranno sia l'accelerazione hardware che i calcoli al livello software. Anche qui, se non è presente l'hardware 3D la chiamata alla funzione fallirà.

Questi flag si escludono a vicenda, ma possono essere combinati con i seguenti altri flag per passare informazioni aggiuntive a DirectX sulla creazione del device:

FPU_Preserve: Questa opzione informa DirectX di effettuare tutti i calcoli usando la precisione a doppia virgola mobile *(float),* che può rallentare le performance.
MultiThreaded: Questo flag indica a DirectX che si ha bisogno di un *safe multithread enviroment.* Per controllare gli oggetti *kernel* per il *multithreading* sicuro, le performance possono abbassarsi. È bene usare questo flah solo quando strettamente necessario.
PureDevice: Questa opzione si usa solo in combinazione con *HardwareVertexProcessing,* e serve a specificare che tutti i calcoli saranno processati a livello hardware, comprese le trasformazioni matriciali, lo shading, l'illuminazione, la rasterizzazione, ecc. Questa è la scelta migliore, ma purtroppo solo poche schede grafiche offrono questa opzione.

Riguardo l'ultimo parametro per la creazione del device, i parametri di presentazione, è una complessa struttura usata per definire molti dettagli di basso livello. I principali sono:

TEXTURE

Le texture sono semplicemente delle bitmap che vengono disegnate sulla superficie degli oggetti per descriverne l'aspetto, ossia i materiali di cui sono fatti. Ad esempio, per rappresentare un muro andremo semplicemente a disegnare la texture di un mattone tante volte, una dopo l'altra, sopra ad un parallelepipedo.

EnableAutoDepthStencil e **AutoDepthStencilFormat:** Questi membri indicano, rispettivamente, se si vuole usare un depth buffer, e di quale formato questo buffer deve essere. Il depth buffer serve a definire la distanza fra gli oggetti del mondo 3D virtuale e lo schermo, ed è quindi responsabile di quali oggetti verranno disegnati sopra ad altri. Dopo aver creato un device con questa opzione, l'applicazione deve settare la proprietà *ZBufferEnable* del device per il *depth buffering.*
BackBufferCount, BackBufferFormat, BackBufferWidth e **BackBufferHeight:** Questi membri definiscono il numero dei *back buffer* (da 1 a 3), il formato di questi buffer (definiti nell'enumerazione *Format*), e la loro larghezza e altezza. I *back buffer* device sono usati per renderizzare la scena in background mentre la precedente è mostrata sullo schermo, in modo da non mostrare sullo schermo disegni di transizione (nessun disegno parziale è mostrato all'utente).
SwapEffect: Questo flag definisce il tipo di passaggio che deve avvenire fra *back buffer* e *frame buffer.* Può essere di 3 tipi: *SwaptEffect.Discard, SwaptEffect.-Flip, SwaptEffect.Copy.* Discard è la scelta di default, e significa che si lascia a DirectX il metodo più adatto da scegliere. *Flip* significa che ci sarà una sorta di rotazione fra i buffer (non appena un frame buffer è stato mostrato su schermo questo diventa un *back buffer,* e il *back buffer* viene mostrato a schermo come un frame buffer e così via in modo ciclico). *Copy* significa che di volta in volta il *back buffer* sarà copiato sul *frame buffer,* ed è attivabile solo se è presente un solo *back buffer (BackBufferCount).*
Windowed: Se si setta su *true,* significa che si vuole far eseguire l'applicazione in una finestra, su *false,* DirectX lavorerà a full-screen.
DeviceWindowHandle: L'handle della finestra che sarà usata da DirectX come lavagna di disegno. Se si setta su null, DirectX userà la finestra attiva.

DisplayMode

Mentre con il termine adapter ci si riferisce ad un componente hardware e i suoi driver e con il termine device si intende l'oggetto principale usato per accedere ad una specifica finestra e poterci disegnare sopra, con il termine display mode si definiscono gli oggetti della classe *DisplayMode* che mantengono le informazioni sullo stato dello schermo, come larghezza, altezza, refresh rate e un flag che indica come i colori sono controllati dal display. Questo flag può essere:

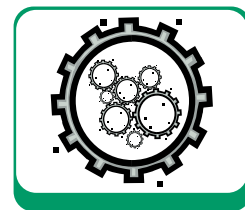
A8R8G8B8: Ogni pixel sullo schermo è definito usando un valore ARGB di 32 bit, dove la componente di rosso, di verde e quella di blu possono assumere un valore da 0 a 255, e la componente A *(Alpha)* rappresenta la trasparenza del pixel (255 è completamente opaco e 0 è interamente trasparente).
X8R8G8B8: Formato di colore sempre di 32 bit, dove però la componente *X* non è usata. Come il precedente è possibile avere fino a 16 milioni di colori.
R5G6B5: Questo formato usa 16 bit, dove ogni componente di colore RGB può assumere 32 differenti valori, più un ulteriore bit per il verde che può assumere fino a 64 possibili valori. Si possono così generare fino a 64 mila colori.
X1R5G5B5: Formato di 16 bit, dove ogni componente può assumere 32 diversi valori, arrivando così ad un totale di 32 mila colori.

Quando si effettua la scelta sul tipo di formato da utilizzare, è importante considerare l'effettiva necessità di colori sullo schermo, per non consumare inutilmente memoria. I formati a 32 bit impiegano quasi il doppio a mostrare lo stesso numero di pixel sullo schermo. È bene notare che se si esegue l'applicazione non a full-screen ma in una finestra, bisognerà usare la risoluzione correntemente impostata del computer. A questo punto siamo in grado di capire il seguente codice necessario per l'inizializzazione di DirectX

```
public bool InitializeGraphics(){
  try {
    // Istanziare i parametri di presentazione e il display
                                                     mode
    PresentParameters presentParams = new
                                    PresentParameters();
    // Non eseguire l'applicazione a full screen
    presentParams.Windowed = true;
    // Lasciamo scegliere a DirectX lo SwapEffect che
                                        ritiene opportuno
    presentParams.SwapEffect = SwapEffect.Discard;
    // Creiamo il device sulla corrente finestra
    device = new Device(Manager.Adapters.Default.Adapter,
                    DeviceType.Hardware, this, CreateFlags.
                SoftwareVertexProcessing, presentParams);
    return true;
    }
    catch {
      return false;
    }
  }
```

CONCLUSIONI

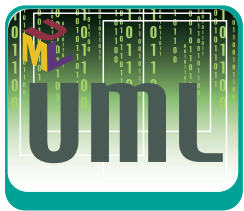
In questo articolo abbiamo conosciuto i mattoni fondamentali di DirectX e della programmazione grafica tridimensionale su Windows. Abbiamo inoltre capito come lavorare con Direct3D e cosa è necessario fare per inizializzarlo correttamente in una applicazione .NET. Nel prossimo affronteremo tutti i passi necessari per poter finalmente iniziare a disegnare oggetti tridimensionali.

Carlo Federico Zoffoli

L'AUTORE

Carlo Federico Zoffoli ha iniziato a scrivere programmi e giochi per PC fin dall'età di 12 anni quando è rimasto affascinato dal film TRON della Walt Disney e le sue esperienze professionali includono lo sviluppo di software multimediali per RAI e MediaSet. Attualmente lavora in I.T. Telecom per OMS, il prodotto per la gestione dei clienti in Outsourcing di Telecom Italia. Per contatti e feedback su questo articolo potete scrivere a:
CarloZoffoli@ioprogrammo.it

La descrizione della dinamica interna alle applicazioni

I sequence diagrams

La vita di un'applicazione è costruita sulla base delle interazioni fra gli oggetti che la compongono. I diagrammi di cui parleremo questo mese servono proprio a descrivere la dinamica interna alle applicazioni.

LINK

Il sito ufficiale del linguaggio UML:
www.uml.org

Il sito del Object Management Group:
www.omg.com

La sezione UML sul sito di Rational Software:
www.rational.com/uml

Negli articoli precedenti abbiamo conosciuto il linguaggio di modellazione e progettazione UML, dedicato alla progettazione di sistemi software di qualità indipendentemente dal linguaggio e dal processo di sviluppo utilizzati. In particolare ci siamo concentrati prima sulla comprensione degli *Use Case Diagram*, che consentono di descrivere uno scenario concreto di operatività degli utilizzatori di un qualunque sistema, e sui *Class Diagrams*, che invece consentono di progettare un sistema software dal punto di vista delle classi oggetti che lo compongono. Sia gli *Use Case Diagrams* che i *Class Diagrams* appartengono alla cosiddetta *Area Strutturale* del linguaggio UML, sono diagrammi, cioè, che consentono la definizione di particolari aspetti statici del modello. UML però comprende anche alcuni diagrammi che si collocano nella cosiddetta *Area Dinamica* e che consentono di modellare i comportamenti degli oggetti anche in relazione al tempo di esecuzione delle singole attività che questi descrivono.
In questo articolo cercheremo di capire come si legge e come si scrive uno di questi diagrammi, il diagramma di sequenza o *Sequence Diagram*.

UN PRIMO ESEMPIO

Un Sequence Diagram è la rappresentazione grafica di uno scenario che descrive le interazioni tra oggetti in funzione di una sequenza temporale. In Fig. 1 possiamo vedere un esempio minimale di un *Sequence Diagram*, nel quale compare l'attore principale dell'interazione che utilizza uno alla volta, direttamente o indirettamente, tutti gli oggetti che concorrono a formare la sequenza.
Guardando l'esempio possiamo subito notare due cose fondamentali:

Fig. 1: Un semplice Sequence Diagram.

- la barra verticale che parte dall'attore e si estende verso il basso delinea la sequenza temporale dell'utilizzo degli oggetti da parte dell'attore stesso;
- gli oggetti sono ordinati in funzione del loro momento di utilizzo, a sinistra quelli utilizzati per primi, a destra gli altri, indipendentemente da chi li utilizza realmente, cioè indipendentemente dal fatto che siano utilizzati direttamente dall'attore oppure da qualche altro oggetto.

La "sequenza" che identifica la nomenclatura *Sequence Diagram* ha dunque un doppio significato, da un lato l'ordine di utilizzo degli oggetti (in orizzontale), dall'altro lo scorrere del tempo dall'alto verso il basso. Nel caso specifico il nostro *Sequence Diagram* descrive, in quest'ordine, i seguenti compiti:

- l'attore utilizza l'oggetto 1 mandandogli un messaggio (identificato nel diagramma da una freccia piena verso destra);
- l'oggetto 1, all'interno del suo periodo di vita, utilizza l'oggetto 2 mandandogli un messaggio;
- l'oggetto 2 esegue il suo compito e restituisce l'esecuzione al chiamante, questo viene indicato attraverso l'utilizzo (facoltativo) di una freccia inversa tratteggiata;
- l'oggetto 1, terminato il suo compito, restituisce l'esecuzione al chiamante;
- l'attore, terminato l'utilizzo dell'oggetto 1, inizia un nuovo passo consistente nell'utilizzo diretto dell'oggetto 2, questo viene fatto mandando un messaggio ad esso;
- l'oggetto 2 termina l'esecuzione e, come di consueto, restituisce al chiamante il controllo dell'esecuzione.

Quella che abbiamo descritto è una sequenza di operazioni che assomigliano molto a quelle contenute normalmente in una funzione di un qualsiasi linguaggio di programmazione oppure in un metodo di un linguaggio ad oggetti. In effetti il primo obiettivo dei *Sequence Diagram* è il permettere la progettazione ed il design di componenti applicati-

ve semplici e concrete, come le funzioni o i metodi.

RELAZIONI TRA CODICE SORGENTE E DIAGRAMMA

In Fig. 2 possiamo vedere un altro esempio di *Sequence Diagram*, questa volta si tratta di diagramma leggermente più complesso. Proviamo ad interpretarlo per renderci conto di cosa si tratta e proviamo a pensarlo come un metodo di una qualsiasi classe Java. Innanzi tutto, se si tratta di un metodo, il nome del metodo è certamente *datePrint*, non riceve parametri e non restituisce valori. Queste informazioni possiamo ricavarle guardando il primo messaggio spedito dal nostro attore. Ricordiamoci infatti che l'attore del diagramma è l'utilizzatore della sequenza di operazioni descritte nel diagramma, pertanto è lui che effettua la chiamata, in questo caso, del metodo. Bene, guardiamo adesso il nome del primo oggetto che viene utilizzato, si tratta della classe *Utils*, alla quale il nostro attore manda il messaggio *datePrint():void*, questo significa, se vogliamo interpretarlo in linguaggio Java, che la classe *Utils* possiede un metodo *datePrint()* che non riceve parametri e non restituisce nulla ed il nostro attore, avendo già a disposizione un'istanza della classe (forse perché astratta, ma questo non lo sappiamo) esegue l'invocazione del metodo. Molto bene, abbiamo capito la struttura generale del metodo e come questo si colloca all'interno di una ipotetica gerarchia di classi.

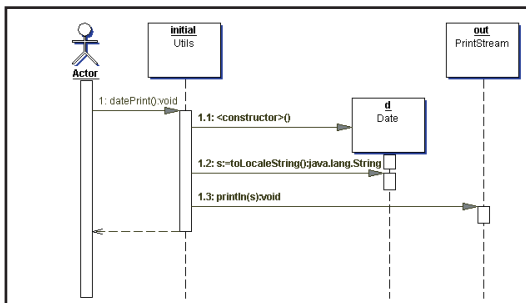



Fig. 2: ***La generazione di nuovi oggetti descritta nel diagramma.***

Guardiamo adesso il contenuto del metodo che nel diagramma è rappresentato con le operazioni che questo svolge all'interno del suo ciclo di vita, cioè come dicevamo prima, tutte le operazioni che partono dalla barra verticale che si trova immediatamente sotto l'oggetto stesso. La prima cosa che viene fatta all'interno del metodo è utilizzare un oggetto *d*, istanza della classe *Date*. La cosa interessante però è che questa non è un'istanza già esistente, si tratta di un oggetto istanziato in questo momento dal metodo attraverso il richiamo al costruttore dell'oggetto *Date*. Possiamo accorgerci di questo in due modi:

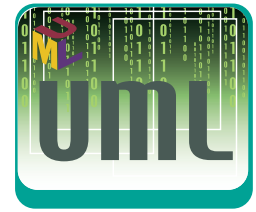
- il messaggio che viene spedito alla classe *Date* è *constructor()* che rappresenta il costruttore dell'oggetto che stiamo utilizzando e di conseguenza verrà creata una nuova istanza della classe;
- l'oggetto che utilizziamo non è allo stesso livello di altezza degli altri, ma è posizionato più in basso nella scala temporale, ad indicare che il suo ciclo di vita inizia in un momento successivo rispetto agli altri ed in particolare inizia soltanto in relazione all'invioo del messaggio stesso, cioè all'esecuzione del costruttore.

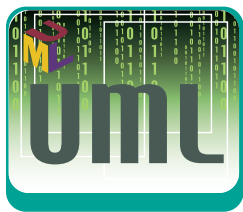
A questo punto abbiamo istanziato un nuovo oggetto, proseguiamo nella scala temporale per vedere quale messaggio gli viene mandato, si tratta di: *s:={ toLocaleString():java.lang.String}*. Il significato di questo messaggio è il seguente: viene chiamato il metodo *toLocaleString()* dell'oggetto *d* ed il suo valore ritornato, (che è un oggetto istanza della classe *java.lang.String*), viene memorizzato all'interno di un oggetto *s* che naturalmente è di tipo compatibile, si tratta infatti di un altro *java.lang. String*. La terza ed ultima operazione compiuta nel metodo consiste nell'utilizzare un'istanza preesistente dell'oggetto *out*, istanza della classe *PrintStream*, per invocare il suo metodo *println*. Questo comportamento viene schematizzato attraverso l'inserimento nel diagramma dell'oggetto in questione, notate però che questa volta l'oggetto è posizionato bene in alto ad indicare che l'istanza era precedente all'esecuzione del metodo stesso. Il messaggio che viene poi spedito a questo oggetto è il seguente: *{println(:java.lang. String):void}(s)*. Si tratta di un metodo che non restituisce alcun valore, infatti è dichiarato *void*, e che riceve il parametro s che era stato popolato in precedenza. Questa è l'ultima operazione eseguita all'interno del metodo. Se provassimo ad utilizzare tutte queste informazioni per scrivere un vero metodo in Java otterremmo il codice seguente:

```
public void datePrint() {
    Date d = new Date();
    String s = d.toLocaleString();
    System.out.println(s); }
```

che è esattamente il codice che implementa il *Sequence Diagram* che abbiamo disegnato.

CREAZIONE ED ELIMINAZIONE DI OGGETTI

Nell'esempio precedente abbiamo visto come è possibile, all'interno di un *Sequence Diagram*, indicare come costruire un nuovo oggetto. Adesso, contrariamente a quanto fatto in precedenza, partiamo da questo codice sorgente:

IL CONSORZIO OMG

OMG (Object Management Group) è il comitato di standardizzazione delle tecnologie ad object oriented.
Tutti gli standard finora approvati dall'ente di standardizzazione sono poi diventati standard de facto, questo è indice della qualità della procedura di standardizzazione.
Nel Gennaio del 1997 la versione 1.0 di UML viene offerta per la standardizzazione al consorzio OMG. Poco più tardi, esattamente il 14 Novembre dello stesso anno, OMG adotta la versione 1.1 come standard.
La notazione prende piede a partire dalla versione 1.3; oggi UML è sicuramente la metodologia di analisi e design ad oggetti più diffusa ed adottata in ambito professionale.

UML E LE AZIENDE

Qui di seguito l'elenco dei contributor che hanno adottato come standard UML, i quali provvedono alla sua divulgazione e che hanno aderito e tuttora contribuiscono al suo sviluppo: Accenture, Ericsson, Hewlett-Packard , I-Logix, IBM, ICON Computing, Intellicorp, ISE, MCI Systemhouse, Microsoft, ObjectTime, Oracle Platinum Technology, Computer Associates, PTech, Rational Software, Reich Technologies, Softeam, Sterling Software, Taskon, Texas Instruments , Unisys.

```
public void datePrint2() {
    Date d = new Date();
    String s = new String();
    s = d.toLocaleString();
    d.finalize();
    System.out.println(s);
    s.finalize();}
```

Si tratta, come si può vedere, di una situazione molto simile a quella precedente, questa volta però gli oggetti vengono tutti costruiti e distrutti esplicitamente. Il *Sequence Diagram* che descrive questo metodo è visibile in Fig. 3. Dopo una prima occhiata ci si può immediatamente rendere conto che, come ci si può aspettare, i costruttori si occupano di istanziare i nuovi oggetti *d* ed *s* e questi vengono posti nel diagramma ad un'altezza compatibile con il momento temporale nel quale sono stati istanziati. Al termine del loro utilizzo i due oggetti vengono esplicitamente eliminati attraverso l'utilizzo del metodo *finalize()* e questo comportamento viene indicato, nel diagramma, attraverso l'utilizzo di una croce nera in prossimità della chiusura della loro timeline. Questa informazione aggiuntiva indica che l'oggetto in questione è stato esplicitamente distrutto e dal codice sorgente che lo aveva in precedenza istanziato, senza aspettare interventi automatici di strumenti di *Garbage Collecting* propri, per esempio, di linguaggi come Java.

CICLI

Tutti i linguaggi di programmazione sono dotati di strutture lessicali che consentono di creare cicli e ripetizioni. Un linguaggio come UML, pertanto, non può non comprendere una sintassi visuale per rappresentarle, soprattutto in contesti dinamici come quelli rappresentati dai *Sequence Diagrams*. Nell'esempio di diagramma presente in Fig. 4 si può trovare qualcosa che fino ad ora non abbiamo ancora incontrato, si tratta di un rettangolo grigio parzialmente sovrapposto ad una parte della barra di esecuzione del metodo *printRows*. I rettangoli di questo tipo rappresentano esecuzioni cicliche oppure condizionate da fattori di contorno, in questo caso si tratta di un elemento di iterazione che consente di realizzare un ciclo *for*. La descrizione della tipologia del ciclo è contenuta nel diagramma stesso: si tratta di una scritta posizionata nell'angolo in alto a destra del rettangolo ed il suo contenuto è il seguente:

Fig. 3: *La distruzione di oggetti descritta nel diagramma.*

Fig. 4: *Rappresentazione di un ciclo for.*

```
for(int index=0;index<numrows;index++)
```

Nei *Sequence Diagrams* è possibile, in questo modo, inserire istruzioni dei seguenti tipi: *if, else-if, else, for, while, do-while, try, catch, finally* e per ognuna di queste è possibile indicare la condizione da applicare al costrutto. Nel nostro caso il costrutto è naturalmente un *for* e la condizione di ciclo è quella indicata tra le parentesi tonde. All'interno del ciclo *for* vengono eseguite alcune operazioni, la prima consiste nella trasformazione in stringa del valore numerico che rappresenta l'indice del ciclo, la seconda è un'operazione di stampa analoga a quella dell'esempio precedente. Se proviamo a fare l'esercizio visto in precedenza di ricostruire il codice sorgente Java rappresentato da questo *Sequence Diagram* otteniamo questo codice:

```
public void printRows(int numrows) {
    for (int index=0; index<numrows; index++)
    {System.out.println("Questa è la riga "
                        + Integer.toString(index) + "\n"); }
}
```

COSTRUTTI CONDIZIONALI

Analogamente a quanto accade per i cicli, anche i costrutti condizionali sono ampiamente supportati dalla sintassi dei *Sequence Diagrams*. In Fig. 5 è visibile un diagramma che contiene un costrutto condizionale, si tratta in particolare di un *if-else*. Il diagramma rappresenta il metodo *printMax*, contenuto nella classe *Utils*, che riceve come parametri due interi e deve stampare il valore del numero che ha il valore più grande. Quello che deve fare, quindi, è confrontare il valore dei due parametri per stabilire qual è il maggiore e quindi stamparlo. Il costrutto condizionale è anche in questo caso rappresentato da un rettangolo grigio parzialmente sovrapposto alla barra che rappresenta il ciclo di vita del metodo, in questo caso però i rettangoli grigi sono due, ce n'è infatti uno per il costrutto *if* ed un altro per il costrutto else. I due rettangolo, per far capire che si tratta di due costrutti correlati, sono legati da una riga obliqua che rappresenta la correlazione tra i due blocchi di istruzioni. Anche in questo caso all'interno di ogni blocco è presente un pacchetto di istruzioni da eseguire e si tratta, anche questa volta, dell'utilizzo della funzione di conversione da intero a stringa per poter

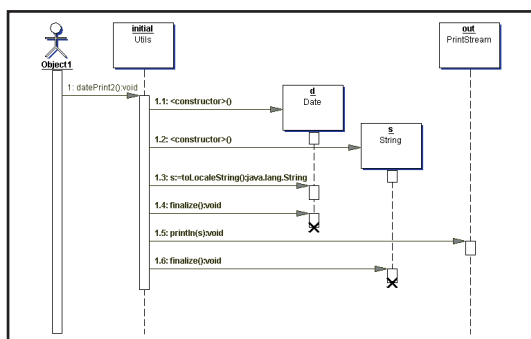
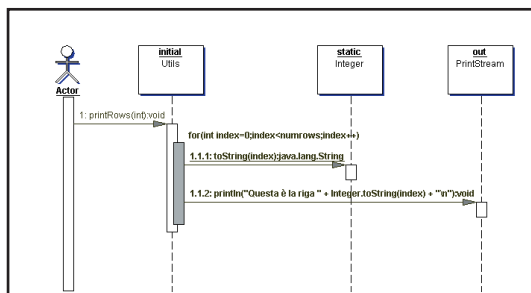
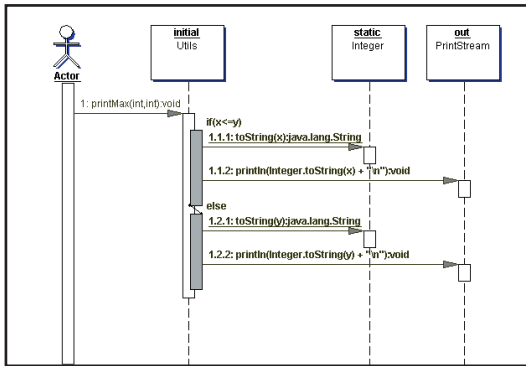



Fig. 5: ***Rappresentazione di un costrutto condizionale.***

garantire la stampa del numero senza errori di conversione e della funzione di stampa vera e propria ottenuta attraverso l'utilizzo del metodo *println* dell'oggetto *out*. Anche in questo caso accanto all'angolo superiore destro del rettangolo più in alto è presente la condizione di discriminazione che consente di decidere se eseguire un blocco di codice oppure l'altro, si tratta della scritta:

```
if(x<=y)
```

Secondo quanto sappiamo non è un problema risalire al codice sorgente rappresentato da questo diagramma, si tratta infatti di questo:

```
public void printMax(int x, int y) {
    if (x<=y)
    { System.out.println(Integer.toString(x) + "\n"); }
    else
    { System.out.println(Integer.toString(y) + "\n");}
}
```

GENERAZIONE AUTOMATICA

Come si è visto in questi esempi esiste una strettissima relazione tra un *Sequence Diagram* ed il codice sorgente in un qualunque linguaggio di programmazione ad oggetti. Questa relazione è talmente forte che in molti tool evoluti di progettazione UML esistono strumenti che consentono di generare il diagramma a partire dal codice sorgente e, dove possibile, di generare il codice sorgente dal diagramma. Nel caso specifico gli esempi di questo articoli sono stati in parte generati in maniera automatica utilizzando *Together ControlCenter* di Togethersoft (recentemente acquisita da Borland). Come visibile in Fig. 6 con *Together ControlCenter* è sufficiente premere il tasto destro del mouse sul nome di un metodo reale già esistente per ottenere una voce del menu contestuale che permette la generazione automatica del *Sequence Diagram* di quel metodo. Allo stesso modo, cliccando con il tasto destro sullo sfondo di un *Sequence Diagram*, compare una voce nel menu contestuale che consente di generare dinamicamente l'implementazione pratica di quanto descritto nel diagramma.

Fig. 6: ***Together a generazione automatica di un diagramma.***

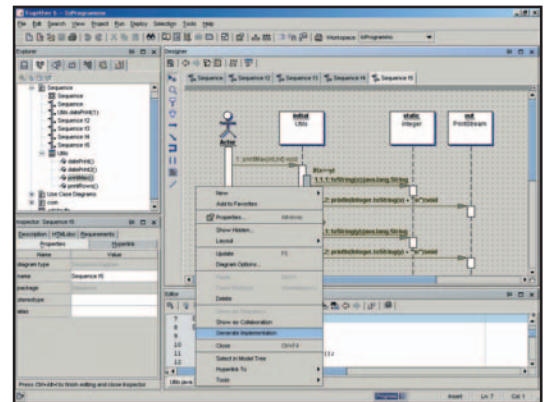
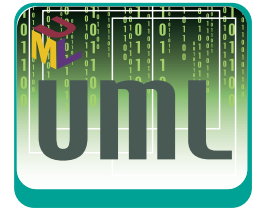



Fig. 7: ***La generazione automatica dell'implementazione di un metodo.***

CONCLUSIONI

Con questa serie di articoli abbiamo iniziato ad apprezzare il linguaggio di modellazione UML per la progettazione di sistemi software di qualità indipendentemente dal processo di sviluppo utilizzato. Abbiamo visto come UML consenta di modellare la realtà attraverso un insieme di diagrammi in grado di occuparsi di ogni singolo aspetto del sistema che si deve modellare. Tra i diagrammi a disposizione abbiamo visto:

- gli **Use Case Diagram**, che hanno l'obiettivo di descrivere uno scenario concreto di operatività degli utilizzatori di un sistema;
- i **Class Diagram**, che consentono di descrivere singolarmente le varie tipologie di oggetti che concorrono a formare il nostro sistema software;
- i **Sequence Diagram,** che consentono di modellare i comportamenti degli oggetti anche in relazione al tempo di esecuzione delle singole attività che questi descrivono.

A questo punto siamo in grado di utilizzare questi potenti strumenti di modellazione e progettazione dei sistemi software per scrivere codice che non sia solo funzionante (come se fosse banale) ma soprattutto sia ben pensato, ben organizzato e ben documentato, in una parola codice di qualità.

Massimo Canducci

Il testo che non può mancare nella biblioteca di chi scrive UML:

• THE UNIFIED MODELING LANGUAGE USER GUIDE
Booch, Jacobson, Rumbaugh
(Addison Wesley) 1999

Altri testi interessanti sull'argomento:

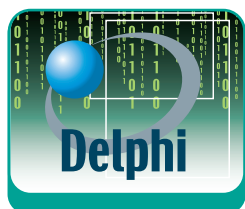
• OBJECT ORIENTED SOFTWARE CONSTRUCTION, 2TH EDITION
Bertrand Meyer
(Prentice Hall) 1997

• VISUAL MODELING WITH RATIONAL AND UML
Terry Quatrani
(Addison Wesley) 1998

Tipi di dato e controllo di flusso

Delphi: corso di Object Pascal

parte seconda

Continuiamo il nostro corso creando un comodo convertitore da una qualunque base numerica al sistema decimale.

L'applicazione di questo articolo è stata creata con la seguente configurazione PC:

Pentium4 2.60GHz, 512Mb RAM

Sistema Operativo: Windows XP Home SP1

Software: Delphi Enterprise 7

Nell'articolo precedente abbiamo iniziato ad esplorare le caratteristiche di Object Pascal, concentrandoci sulla struttura di un programma e sui rudimenti del linguaggio. Abbiamo parlato di unit per la creazione di codice modulare, abbiamo brevemente accennato al concetto di routine ed abbiamo appreso a lavorare con la console di testo. Credo di aver anche, ad un certo punto, definito il Pascal come un linguaggio di programmazione fortemente tipizzato, ed il primo compito che ci poniamo oggi è proprio quello di approfondire questa affermazione, dedicando la nostra attenzione a variabili e tipi di dato, prima di passare a studiare le strutture iterative e condizionali che possiamo utilizzare quando scriviamo codice in Delphi.

I TIPI DI DATO IN PASCAL

Alcuni linguaggi, quali PHP, Perl, JavaScript e simili, permettono di utilizzare variabili senza specificare quale sia la tipologia di informazione che andranno a contenere. Addirittura con una parte di essi è possibile proprio iniziare ad utilizzare una variabile senza neanche dichiararla all'interprete del codice, lasciando così a questo il compito di allocare al volo lo spazio necessario per tali variabili non conosciute, appena queste ultime vengono referenziate. La quantità di spazio dedicato alle variabili in questi linguaggi varia a run-time e viene determinata ogni volta che si modifica il contenuto della stessa.
Sebbene un approccio di questo tipo renda più semplice e veloce la stesura di codice, permettendo modifiche dei propri algoritmi e ripensamenti senza grosse conseguenze sintattiche, lo svantaggio è che tutto il lavoro lasciato all'interprete costa in termini di pesantezza ed efficienza dell'esecuzione. Dall'altro lato, invece, abbiamo linguaggi più simili al Pascal, tra cui Java e C/C++ che richiedono, per poter essere compilati, che tutte le variabili che utilizzeremo siano dichiarate prima di essere referenziate nel codice e che in tale dichiarazione sia esplicitamente ed inequivocabilmente indicato che tipo di dato la variabile potrà contenere: qualunque tentativo di inserimento di un valore di tipo non compatibile con quello indicato risulterà nella non compilazione del codice e nella conseguente impossibilità di eseguirlo. In questo caso, ovviamente, il programmatore è molto più vincolato che in linguaggi a tipizzazione debole o assente, ma è anche vero che risulta più difficile scrivere codice errato o con bachi banali, ma faticosi da rintracciare. Il fatto di dover dichiarare preventivamente le variabili e il loro tipo, ci permette insomma di fare chiarezza sul codice che scriviamo e di essere molto più precisi, innanzi tutto con noi stessi.
Spostando il discorso verso il Pascal, poi, dobbiamo dire che esso è tra i più severi linguaggi tipizzati. Il C/C++ e Java ci consentono infatti di dichiarare le variabili ed il loro tipo in qualunque punto del codice, a patto che la dichiarazione avvenga prima che le variabili stesse vengano inizializzate o referenziate. In Pascal, invece, i punti in cui si possono dichiarare le variabili sono fondamentalmente a livello di ogni *program* (o *unit*) oppure dopo l'intestazione di una routine (*procedure* o *function* che sia). Bisogna quindi avere molto chiaro dall'inizio, di quali variabili si avrà bisogno e di che tipo saranno. Ma vediamo adesso quali sono le tipologie di dati che possiamo associare alle variabili in Pascal.
In qualità di lettori di ioProgrammo, avendo già dimestichezza con altri linguaggi di programmazione, non farete fatica a riconoscere le corrispondenze tra il Pascal e i "concorrenti", per cui

vedremo di creare una piccola mappa verso C/C++, Java e Visual Basic.

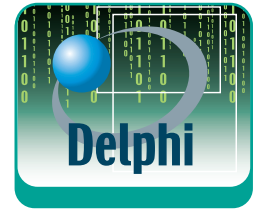
COMINCIAMO DAI NUMERI!

Il Pascal ha un'interessante nozione che è quella dei tipi ordinali, cioè di tipi che possono essere enumerati e tali per cui per ogni valore può sempre essere dato un successivo ed un precedente: per tutti queste tipologie di dato esistono una serie di metodi per manipolare e gestire l'enumerazione, tra cui *Pred(x)* e *Succ(x)* che restituiscono rispettivamente il valore prima e quello dopo di *x*, e *Ord(x)* che restituisce invece la posizione ordinale di *x* nell'enumerazione che si sta utilizzando. Altri due metodi interessanti sono *Inc(x)* e *Dec(x)*, che semplicemente avanzano od indietreggiano di una posizione nell'enumerazione. Nella pratica questi metodi corrispondono ad *x := x + 1 e x := x – 1*, ma è da notare che la loro compilazione è ottimizzata e risulta quindi più efficiente della versione estesa. Esiste anche una direttiva di compilazione (la *{$RANGECHECKS ON}*, vedi Box per maggiori informazioni sulle opzioni di compilazione) che permette di richiedere al compilatore di verificare a run-time che le assegnazioni di valori dentro di un variabile ordinale restino nei limiti del range del tipo stesso, lanciando un'eccezione in caso contrario (delle eccezioni parleremo più avanti, dopo aver trattato della programmazione ad oggetti): questa opzione, però, per quanto comoda e preventiva, ingrandisce un po' il codice generato e ne rallenta l'esecuzione. Senza questa opzione, una volta raggiunto il massimo di un range, si ricomincia dall'inizio. Se, ad esempio, la variabile myvar può contenere valori da 0 a 255 e io eseguo *myvar := 270;* alla fine troverò in *myvar* il valore 15.

I valori interi, in Pascal, sono il tipo ordinale per eccelenza e vengono normalmente rappresentati con un *Integer*, (corrispondente all'*int* di Java e C++ e all'*Integer* di Visual Basic). Si tratta di un valore che può andare da *-2147483648* a *2147483647* nella versione con segno, oppure senza segno – in questo caso il Pascal lo chiama *Cardinal* – da *0* a *4294967295*, coincidendo così con il tipo *unsigned int di C++*. Oltre a questi, abbiamo poi le variazioni sul tema, ovverosia i tipi numerici interi di dimensioni diverse: li riassumo per riferimento nella Tabella 1, dove potete verificare anche la loro capienza. Tra gli altri tipi di dati ordinali in Pascal, troviamo anche i caratteri, i booleani e le enumerazioni. Vediamoli insieme uno per uno!

I *booleani* possono assumere solo uno di due

Tipo	Minimo	Massimo
Integer	-2147483648	-2147483647
Cardinal	0	4294967295
Shortint	-128	127
Byte	0	255
Smallint	-32768	32767
Word	0	65535
Longint	-2147483648	-2147483647
Longword	0	4294967295
Int64	-2^{63}	2^{63-1}

TABELLA 1: I tipi numerici interi del Pascal.

valori (*True* o *False* in ObjectPascal), che sono l'uno il successivo (ed il precedente!) dell'altro. Questo tipo non esiste nativamente in C/C++, mentre si chiama boolean in Java e Boolean in Visual Basic e Pascal.

NOTE

VARIABILI: DICHIARARE, INIZIALIZZARE, REFERENZIARE

Facciamo un poco di chiarezza su alcuni termini utilizzati in riferimento alle variabili. Ciò che diremo vale non solo strettamente per il Pascal, ma per qualunque altro linguaggio.
Innanzitutto le variabili si dichiarano: la dichiarazione è uno statement con il quale si indica al computer con che nome chiameremo una variabile, che tipo di dati dovrà contenere, e conseguentemente quanto spazio dovrà essere allocato per essa. Dopodichè, le variabili devono essere inizializzate, cioè gli deve essere assegnato un valore iniziale affinché non contengano dati random: spesso questa operazione viene fatta in un unico statement con la dichiarazione. Quando una variabile è stata creata ed inizializzata, allora la si può usare utilizzando il suo nome per recuperare il valore che contiene, e questo processo lo chiamiamo referenziazione.

I valori booleani (e solo quelli) possono essere utilizzati nelle espressioni condizionali degli statement *if, while* e *until* (di cui parleremo più avanti in questo articolo). Le enumerazioni sono invece una serie di valori ordinati associati ad identificativi.

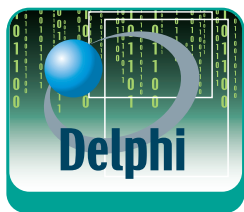
Si tratta effettivamente di tipi di dati personalizzati, per cui è necessario utilizzare la parola chiave *type* per definirne una prima di poterla assegnare come tipo ad una variabile. È così che si creano le enumerazioni, quelle – per esempio – dei semi e delle carte da gioco:

```
type
  Semi = (Cuori,Quadri,Fiori,Picche);
  Carte = (Asso,Due,Tre,Quattro,Cinque,Sei,Sette,
                                  Fante,Donna,Re);
```

per poi assegnarli così ad una variabile

```
var
  carta1seme: Semi;
  carta1val: Carte;
```

NOTA

È possibile scaricabile Delphi in versione di prova dal sito www.borland.com
È richiesta una registrazione gratuita a fronte della quale si riceverà la licenza temporanea di utilizzo del prodotto. Ai fini di questo corso, anche se non state testate, sono sicuramente adatte anche versioni precedenti di Delphi (5 o 6, per esempio).

```
  carta2seme: Semi;
  carta2val: Carte;
```

e poter dare poi i valori dell'enumerazione tramite un'assegnazione in questo modo

```
carta1seme := Quadri;
carta1val := Fante;
carta2seme := Picche;
carta2val := Asso;
```

Infine, i caratteri (alla pari del char di C++ e Java) rappresentano un singolo carattere: i valori letterali di questo tipo di dato si possono codificare come stringhe di lunghezza uno (quindi delimitati da un singolo apice), oppure con il codice numerico ASCII preceduto da # (per esempio, la *"A"* si può indicare come *'A'* o con *#65*). Si crea una variabile di questo tipo con la parola chiave *Char,* che in realtà attualmente è un sinonimo di *AnsiChar,* per differenziare da *WideChar* che invece rappresenta i caratteri Unicode di due byte. Passiamo adesso a dati non ordinali: per quanto riguarda i tipi di valori a virgola mobile, li trovate nella Tabella 2, sempre insieme alle rispettive dimensioni. È da notare che il tipo *Real* in versioni precedenti di *ObjectPascal* era un valore a 48bit e che oggi invece è di 64 bit ed è identico al tipo *Double.* È necessario rivedere il codice vecchio per sostituire i tipi *Real* con *Real48* se volete continuare a lavorare con il formato a 6 byte, oppure attivare l'opzione di compilatore per la compatibilità *Real* con la direttiva *{$REALCOMPATIBILITY ON}* di modo che i *Real* vengano interpretati come *Real48* senza necessità di modificare il codice. Come ciliegina sulla torta abbiamo finalmente le stringhe: anche qui ci sarebbe una distinzione da operare tra i testi ANSI e quelli Unicode, ma il problema – almeno per ora – non ce lo poniamo… per i programmi scritti nel mondo occidentale i caratteri di un byte sono più che sufficienti. Ecco quindi che la parola chiave *string* ci permette di definire una variabile che conterrà una stringa (nella realtà, si tratta di un sinonimo di *AnsiString*). In Pascal le stringhe letterali si indicano tra apici singoli, in maniera del tutto identica ai caratteri, solo che la lunghezza non è limitata ad uno. Una variabile di tipo stringa dà l'accesso ai singoli caratteri di cui è composta tramite le parentesi quadre, ed è da notare che si può anche modificare il carattere così referenziato. Il semplice blocco di codice che segue mostra un esempio di quanto detto convertendo una stringa in maiuscolo carattere per carattere:

```
var
  laMiaStringa: string;
  cnt: Integer;
begin
  ...
  cnt := Length(laMiaStringa);
  while cnt > 0 do
  begin
    laMiaStringa[cnt] := AnsiUpperCase(laMiaStringa[I]);
    cnt := cnt - 1;
  end;
end;
```

Ora, ripassiamo invece la sintassi della dichiarazione di una variabile, ricordando che tutte le variabili in ObjectPascal devono essere rese note al compilatore prima del loro utilizzo nel codice e che questa informazione preventiva va data nella sezione var di un programma, di una unit o di una routine.

```
var
   <nome> : <tipo>;
```

A seconda, poi, di dove viene inserito il blocco di cui sopra, la variabile creata avrà una visibilità diversa: a livello di programma sarà visibile in tutto il codice dell'applicazione in esecuzione, in una unit, invece, dobbiamo distinguere tra *implementation* e *interface:* se il blocco *var* sta nella prima, la variabile sarà utilizzabile solo dentro la *unit,* dove sarà riconosciuta da tutte le routine, mentre se sta nell'interfaccia, sarà visibile – oltre che in tutta la unit – anche in tutto il codice dove l'unità viene usata. Infine un blocco *var* prima del blocco di una routine rende la variabile locale, e quindi disponibile solamente nel codice delle procedura stessa.

NOTA

LE DIRETTIVE DI COMPILAZIONE

Attraverso le direttive di compilazione specifichiamo delle opzioni attraverso cui il compilatore modifica leggermente il processo di conversione del codice sorgente in codice eseguibile venendo incontro a nostre specifiche esigenze. Esistono moltissimi switch di compilazione e per averne un'idea esaustiva è meglio far riferimento alla guida online di Delphi. In ObjectPascal le opzioni vengono passate sotto forma di commenti con parentesi graffa, dentro cui troviamo un $ e l'opzione stessa, ad esempio

```
{$OVERFLOWCHECKS ON}
```

con cui si attivano i controlli sugli overflow nelle operazioni su valori ordinali.

Tipo	Minimo	Massimo
Real48	2.9×10^{-39}	1.7×10^{38}
Single	1.5×10^{-45}	3.4×10^{38}
Double or Real	5.0×10^{-324}	1.7×10^{308}
Extended	3.6×10^{-4951}	1.1×10^{4932}
Currency	-922337203685477.5808	922337203685477.5807

TABELLA 2: I tipi numerici a virgola mobile di ObjectPascal.

LA MODIFICA DEL FLUSSO ESECUTIVO

Il Pascal dispone di strutture di controllo molto potenti e simili a quelle del C++, di Java, di Visual Basic e altri linguaggi moderni.
Due sono quelle condizionali: *if,* per testare una condizione ed agire in base ad un risultato vero o falso, e *case,* utile invece per diramare l'esecuzione in diversi canali a seconda del valore contenuto nella variabile testata. La sintassi dei due statement è la seguente

```
if <cond> then
  <statement>;
oppure
if <cond> then
  <statement>
else
  <statement>;
ed infine
case <var> of
  <val1>: <statement(s)>;
  <val2>: <statement(s)>;
  …
  <valn>: <statement(s)>;
  [else <statement(s)>;]
end;
```

La funzione degli statement è identica ai corrispondenti if di C++, Java e *If* di Visual Basic, e *switch* di C++, Jave e *Select Case* di Visual Basic. Notate che per *<statement>* si intende o un singolo statement o un blocco *begin…end,* mentre per *<statement(s)>* non è necessario creare un blocco, più comandi possono susseguirsi delimitati dal punto e virgola. Osservate anche come nell'if sia errato porre un punto virgola nello statement prima dell'eventuale *else.*
Passiamo invece ai costrutti iterativi, quelli cioè che consentono di ripetere blocchi di codice selettivamente ed un tot numero di volte. In C++, Java e Visual Basic abbiamo il *for* (*For* in VB) che ci consente di eseguire un numero noto di volte una sequenza di comandi: chiaramente questo non poteva mancare in Pascal ed ha la seguente forma

```
for <indice> := <inizio> to <fine> do
  <statement>;
```

in cui *<statement>* (che può essere un blocco *begin…end* o un singolo comando) verrà eseguito *<fine>* - *<inizio>* volte con *<indice>* che aumenta di 1 ad ogni passaggio andando da *<inizio>* a *<fine>*.
Questa seconda versione invece

```
for <indice> := <inizio> downto <fine> do
  <statement>;
```

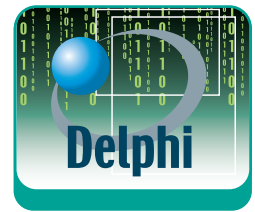
funziona al contrario, diminuendo di 1 il valore di *<indice>* ad ogni iterazione assumendo che *<fine>* sia minore di *<inizio>*.
Gli altri due costrutti di ciclo che offre il Pascal sono invece guidati da una condizione di permanenza *(while)* o uscita *(until)* dal ciclo, e non è quindi determinabile a priori il numero di volte che il codice verrà eseguito. La differenza principale tra *while* e *until* è che la condizione di ripetizione del ciclo è valutata nel primo caso preventivamente all'esecuzione della prima iterazione, e può di fatto verificarsi che il ciclo non venga mai eseguito, mentre con *repeat…until* sappiamo che almeno un'iterazione verrà senz'altro portata a termine. Ecco la sintassi dei due statement:

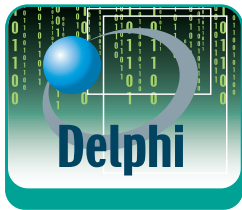
```
while <condizione> do
  <statement>;

repeat
  <statement(s)>
until <condizione>;
```

Notate che anche qui per il *repeat* – come per il case poco sopra – non è necessario creare blocchi *begin…end* quando abbiamo più di un comando nel ciclo. Inoltre nel caso del *while* la condizione indica permanenza nel ciclo (fintantoché *<condizione>* è vera, cicla), mentre nel *repeat* la condizione è di uscita (appena *<condizione>* diventa vera, esci dal ciclo).
A mo' d'esempio, il *Listato 1* ed il *Listato 2* contengono quella parte del codice dei programmi di prova più disseminata di cicli e condizioni.

```
Listato 1:

function ConvertToDec(val: string; base: Integer):
                                          Integer;
var
  idx: Integer;
  cur: Integer;
  res: Integer;
begin
  if (base > 36) or (base < 2) then
  begin
    Result := -1;
    Exit;
  end;
  res := 0;
  cur := 0;
  for idx := Length(val) downto 1 do
  begin
    if val[idx] in ['0'..'9'] then
      cur := ord(val[idx]) - ord('0')
    else if val[idx] in ['a'..'z'] then
      cur := ord(val[idx]) - ord('a') + 10
    else if val[idx] in ['A'..'Z'] then
```

I TIPI INTERI DI RIFERIMENTO

Il tipo *Integer* insieme al tipo *Cardinal,* in Pascal, sono un po' particolari perché la loro dimensione dipende dal sistema su cui gira il compilatore. Su macchine a 16bit (ne esistono ancora?) l'Integer occupa due byte, mentre con processori a 32bit lo stesso tipo compila occupando quattro byte. È facile immaginare che sui compilatori che saranno scritti appositamente per i nuovi processori a 64 bit gli interi di riferimento (come vengono chiamati questi tipi a dimensione non certa) finiscano con valere otto byte! Una cosa simile avviene anche in C/C++, Visual Basic, etc.

```
    cur := ord(val[idx]) - ord('A') + 10
  else
  begin
    Result := -2;
    Exit;
  end;
  if (cur > (base - 1)) then
  begin
    Result := -3;
    Exit;
  end;
  res := res + cur * Floor(Power(
                         base,Length(val)-idx));
 end;
 Result := res;
end;
```

Nel resto del codice allegato alla rivista per questo articolo troverete anche altri esempi pratici di utilizzo delle strutture appena discusse.
Le due applicazioni di questo numero utilizzano una unità che trovate nella cartella *\common* e che contiene la logica di conversione da una base numerica a quella decimale (il metodo più importante è quello del *Listato 1*).
Ricordate che nello scrivere una funzione in Pascal, per poter ritornare un valore al chiamante, si utilizza la variabile predefinita *Result*, o una variabile creata automaticamente che ha lo stesso nome della funzione. L'ultima riga di *ConvertToDec* poteva quindi anche essere:

```
ConvertToDec := res;
```

Questa funzione converte un numero nella base data in un numero in base dieci ed è una funzione privata dell'unità (è solo definita nell'implementazione). Nell'interfaccia, invece, ci sono altre funzio ni che usano *ConvertToDec* per convertire da esadecimale, ottale e binario, e tali funzioni vengono

```
Prompt dei comandi
D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc
Errore di sintassi
MyCalc "<expr>"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "&FF-$177+#1020"
Errore di sintassi
Not a valid operator [2]

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "&FF-$177+#100"
The result of "&FF-$177+#100" is "132"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "12A8/2-4*3"
Errore di sintassi
Not a valid operator [A]

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "128/2-4*3"
The result of "128/2-4*3" is "180"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "#100^#10"
The result of "#100^#10" is "16"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "2456%7"
The result of "2456%7" is "6"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>mycalc "2456/7"
The result of "2456/7" is "350"

D:\WORK\Edizioni Master\CorsoDelphi\lezione2\prjCalc1>
```

Fig. 1: ***Utilizzo dell'applicazione testuale per eseguire calcoli aritmetici.***

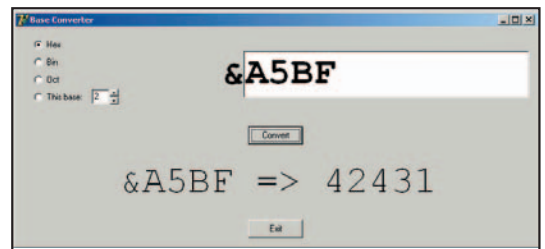



Fig. 2: ***L'applicazione grafica al lavoro (conversione esadecimale).***

richiamate dalle applicazioni fornite come esempio. Delle due, *MyCalc* è un programma di console che interpreta la riga di comando per calcolare delle espressioni matematiche che possono contenere numeri in base 16, 8 e 2 (vd. Fig. 1): non valgono le regole di priorità algebriche, per cui le varie operazioni vengono banalmente eseguite nell'ordine di apparizione.
Nell'espressione immessa come parametro dell'eseguibile (usate le virgolette!) potete usare gli operandi classici + - * / oltre a % (resto di una divisione) e ^ (elevamento a potenza), mentre una serie di caratteri arbitrari (li ho scelti in base al mio gusto personale) identificano un numero esadecimale *(&)*, ottale *($)* e binario *(#)*.
La seconda applicazione è grafica, invece (vd. Fig. 2 e Fig. 3) ed offre un'interfaccia per la conversione di numeri.

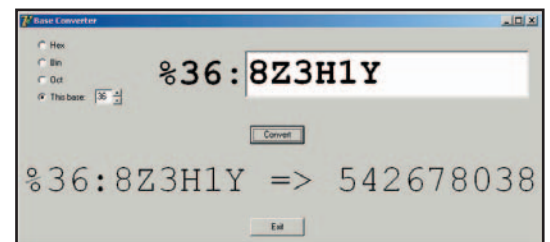



Fig. 3: ***L'applicazione grafica al lavoro (conversione in base 36).***

Si può usare una qualunque base numerica da 2 a 36, dove per basi superiori a 10 vengono utilizzate le lettere dalla *A* alla *Z*. Come sempre non mi soffermo sull'aspetto della creazione e gestione di finestre, in quanto non è obiettivo primario del nostro corso, che si concentra solo sulle caratteristiche del linguaggio.

CONCLUSIONI

Nel prossimo numero andremo ad aggiungere alcune feature a *MyCalc*, complicando le capacità del nostro parser di espressioni algebriche grazie all'uso più consapevole di funzioni e procedure e delle variabili statiche.
Vi aspetto dunque tutti tra un mese!

Federico Mestrone

Gli elementi di base di qualsiasi applicazione

I segreti per manipolare numeri, stringhe e date

In questo appuntamento scopriremo assieme tutto quello che volevate sapere per lavorare senza problemi con dati numerici, stringhe di caratteri e date.

L'obiettivo di questo articolo, è di fornirvi gli strumenti per gestire tutte le problematiche che si presentano quando si deve operare con i tipi di dati fondamentali del linguaggio. Ci occuperemo di descrivere in modo dettagliato gli operatori e le funzioni, che permettono di eseguire calcoli e di manipolare numeri, stringhe e date.
Visual Basic fornisce tre tipi di operatori

- **Aritmetici**
- **Logici**
- **Di confronto**

GLI OPERATORI ARITMETICI

Gli operatori aritmetici permettono di usare il computer come una calcolatrice. VB supporta i quattro operatori aritmetici fondamentali: *Addizione(+), Sottrazione(-), Moltiplicazione(*), Divisione (/).*
Per sommare due variabili numeriche ed assegnare il risultato dell'operazione ad un'altra variabile *Risultato*, si può scrivere:

```
Dim x As Integer
Dim y As Integer
Dim Risultato As Integer
x = 14
y = 45
Risultato = x + y  'Risultato conterrà il valore 59
```

Con le prime tre istruzioni vengono dichiarate tre variabili, nelle successive due istruzioni vengono assegnati i valori alle variabili *x* ed *y*, e nell'ultima istruzione viene assegnato alla variabile *Risultato* la somma di *x* ed *y*. Lo stesso procedimento vale per gli altri operatori. In VB è disponibile l'operatore di divisione a numero intero (\) che viene eseguito più rapidamente, ma restituisce, appunto, un risultato intero troncando la parte decimale.

Naturalmente per entrambi gli operatori "/" e "\", dividendo un numero per zero si solleva un'eccezione di Overflow. L'operatore *Mod* restituisce il resto di una divisione tra due numeri, spesso tale operatore viene usato per testare se un numero è multiplo di un altro numero, poiché in questo caso il risultato è zero.

```
If x Mod y = 0 Then
    Multiplo = True
Else
    Multiplo = False
End If
```

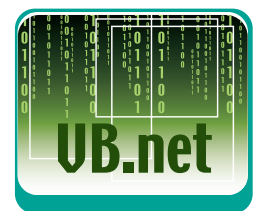
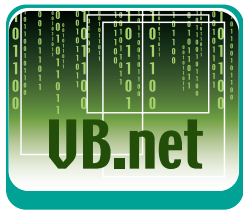
VB supporta l'operatore esponenziale (^) che calcola l'*n*-esima potenza di un numero. Il risultato è dato dall'elevazione del primo operando alla potenza indicata dal secondo. Questo operatore restituisce in ogni caso un risultato di tipo *Double*. In VB.Net tutti i metodi e le costanti necessarie per eseguire funzioni trigonometriche, logaritmiche e normali funzioni matematiche sono racchiuse nella classe *Math*. La classe *Math* rende disponibili numerose funzioni.

Sqrt	Per calcolare la radice quadrata di un numero
Abs	Restituisce un valore che specifica il valore assoluto di un numero.
Exp	Per calcolare e (base dei logaritmi naturali) elevato a potenza.
Log	Per calcolare il logaritmo naturale di un numero.
Log10	Restituisce il logaritmo di base 10 del numero specificato.
Sin, Cos, Tan	Le funzioni trigonometriche che restituiscono rispettivamente: il seno, il coseno, e la tangente di un numero
Asin, Acos, Atan	Le funzioni trigonometriche che restituiscono rispettivamente l'angolo il cui seno, coseno, e tangente è il numero specificato

TABELLA 1: Le funzioni della classe Math.

GLI OPERATORI LOGICI

Gli operatori logici restituiscono i valori booleani *True* e *False* e vengono solitamente usati, insieme all'istruzione *If…Then…Else,* per prendere delle decisioni all'interno del codice. Gli operatori più

comuni sono: *And, Or, Not* e *Xor.* Gli operatori *And, Or, Xor* confrontano due valori *True* o *False* e restituiscono un nuovo valore Booleano calcolato.

```
BelTempo = SolePresente And TemperaturaMite
```

La variabile *Beltempo* sarà pari a *true* soltanto se sono *True* sia *SolePresente* sia *TemperaturaMite*

```
BelTempo = SolePresente Or TemperaturaMite
```

La variabile *Beltempo* sarà pari a *True* quando una qualsiasi delle due variabili *SolePresente* e *TemperaturaMite* sono *True*

```
BelTempo = SolePresente Xor TemperaturaMite
```

La variabile *Beltempo* sarà pari a *True* quando soltanto una (e non tutte e due) delle variabili *SolePresente* e *TemperaturaMite* sono *True.* L'operatore *Not Modifica* un valore *True* in un valore *False* e viceversa. Per maggiore chiarezza, di norma, si utilizzano le parentesi:

```
BruttaGiornata = Not (BellaGiornata)
```

GLI OPERATORI DI CONFRONTO…

Gli *Operatori di Confronto* vengono utilizzati per confrontare operandi sia numerici sia stringa:

> **Maggiore di**
>= **Maggiore o uguale a**
< **Minore di**
<= **Minore o uguale a**
= **Uguale a**
<> **Diverso da**

Il risultato sarà un valore Booleano pari a *True* se la condizione di confronto è soddisfatta, oppure pari a *False* se la condizione di confronto non è soddisfatta:

```
Dim MiaEtà As Integer
Dim EtàMinima As Integer
Dim PossoVotare As Boolean
MiaEtà = 25
EtàMinima = 18
If MiaEtà > EtàMinima Then
    PossoVotare = True
Else
    PossoVotare = False
End If
MessageBox.Show(PossoVotare) 'visualizza True
```

Tutti gli operatori (anche di tipo diverso) possono essere usati in combinazione tra loro, in questo caso il compilatore valuta e risolve ciascuna parte dell'espressione secondo un ordine prestabilito (vedi box). Per evitare l'ordine di precedenza degli operatori, e fare in modo che parti di un'espressione vengano eseguite prima di altre, è possibile utilizzare le parentesi tonde. Gli operatori tra parentesi tonde hanno sempre la precedenza rispetto agli altri, ed in ogni caso all'interno delle parentesi tonde, vengono rispettate le regole di precedenza.

… DI CONCATENAZIONE DI STRINGHE

L'operatore *"&"* viene utilizzato per concatenare due stringhe, il risultato è una stringa formata dai caratteri della prima stringa seguiti dai caratteri della seconda:

```
MsgBox "Prima" & "Vera"  'Visualizza "PrimaVera"
```

Se uno dei due operandi non è di tipo *stringa,* viene automaticamente convertito in stringa. Per concatenare due stringhe è, inoltre, possibile utilizzare l'operatore +, ma in questo caso non è sempre possibile stabilire se verrà eseguita un'addizione o una concatenazione di stringhe. Per eliminare qualsiasi ambiguità, è quindi preferibile utilizzare l'operatore &. In molti casi una stringa contiene degli spazi vuoti che possono indurre in errori difficili da rintracciare all'interno del codice. Per eliminare spazi vuoti all'inizio o alla fine di una stringa si possono utilizzare le funzioni:

- **LTrim** Elimina gli spazi iniziali
- **RTrim** Elimina gli spazi finali
- **Trim** Elimina gli spazi iniziali e finali

```
Dim Testo As String
Testo = "  Ciao  " 'contiene due spazi a sinistra e due
                                              spazi a destra
MessageBox.Show(LTrim(Testo)) ' Visualizza "Ciao  "
MessageBox.Show(RTrim(Testo)) ' Visualizza "  Ciao"
MessageBox.Show(Trim(Testo))  ' Visualizza "Ciao"
```

Per determinare la lunghezza di una stringa (spazi compresi) si utilizza la funzione *Len.* La funzione *Len* restituisce un valore di tipo *Integer* pari al numero di caratteri di una stringa, per l'esempio precedente *MessageBox.Show(Len(Testo))* visualizza *"6".* Per convertire una stringa in caratteri minuscoli o maiuscoli si possono usare le funzioni *Lcase* ed *Ucase.* La funzione Lcase converte la stringa specificata in caratteri minuscoli. La funzione *Ucase* converte la stringa specificata in caratteri maiuscoli.

```
MessageBox.Show(LCase("Ciao a Tutti"))
```

```
'Visualizza "ciao a tutti"
MessageBox.Show(UCase("Ciao a Tutti"))
'Visualizza "CIAO A TUTTI"
```

PRELEVARE CARATTERI DA UNA STRINGA

Per estrarre un numero specificato di caratteri di una stringa a partire da sinistra, si può usare la funzione *Left*. Per estrarre un numero specificato di caratteri di una stringa a partire da destra, si può usare la funzione *Right*. Per entrambe le funzioni la sintassi è: *funzione(stringa, lunghezza)*. Il primo argomento è costituito dalla stringa che si vuole scomporre, il secondo argomento specifica il numero di caratteri che devono essere restituiti. Se il valore di lunghezza è pari a zero il risultato sarà una stringa di lunghezza zero ("").

```
Dim Testo As String
Testo = "Ing. Massimo Autiero"
MessageBox.Show(Microsoft.VisualBasic.Left(Testo, 4))
'Visualizza "Ing."
MessageBox.Show(Microsoft.VisualBasic.Right(
Testo, 7)) 'Visualizza "Autiero"
```

Se *Left* e *Right* vengono utilizzate nelle *Windows Form* o in qualsiasi altra classe che comprende una proprietà *Left* o *Right*, è necessario qualificarle in modo completo facendole precedere dalla classe *Microsoft.VisualBasic*. Entrambe le funzioni possono essere usate in combinazione con la funzione *Len*. Ad esempio per eliminare i primi tredici caratteri di una stringa:

```
MessageBox.Show(Microsoft.VisualBasic.Right(
Testo, Len(Testo) - 13))
'Visualizza "Autiero"
```

Ancora più interessante è la funzione *Mid*, tale funzione permette di estrarre una stringa a partire da un punto qualsiasi della stringa d'origine (e non per forza da un estremo come per le due funzioni precedenti). La sintassi è:

```
Mid(stringa, inizio[, lunghezza])
```

In cui *inizio* rappresenta la posizione iniziale da cui si deve partire per estrarre la stringa, e lunghezza è un parametro opzionale che indica la lunghezza della stringa risultato. Se il parametro lunghezza viene omesso verranno restituiti tutti i caratteri a partire dalla posizione indicata in inizio.

```
MessageBox.Show(Mid(Testo, 6, 7)) 'Visualizza "Massimo"
```

La funzione *Mid* può essere usata anche come comando, poiché permette di sostituire parte di una stringa con un'altra stringa.

```
Mid(Testo, 6, 7) = "Antonio"
MessageBox.Show(Testo) 'Visualizza "Ing. Antonio Autiero"
```

Per rintracciare una sottostringa in una stringa più grande si può utilizzare la funzione *InStr*. La funzione *InStr* restituisce un valore intero che indica la posizione d'inizio della prima occorrenza di una stringa all'interno di un'altra. La sintassi è:

```
InStr([inizio, ]stringa1, stringa2[, tipodiconfronto])
```

In cui: inizio è un parametro facoltativo che indica la posizione a partire dalla quale si deve iniziare la ricerca di *stringa2*. Se inizio viene omesso la ricerca inizia dal primo carattere. stringa1 è la stringa in cui ricercare *stringa2*. tipodiconfronto è un parametro facoltativo che permette di specificare il tipo di confronto delle stringhe. La funzione *InStr* restituisce un valore pari a zero se *stringa2* non è contenuta in *stringa1*. Interessante è la funzione *Replace* che permette di trovare e sostituire una sottostringa con un'altra sottostringa. La sintassi è più complessa delle altre funzioni:

```
Replace(stringa, stringadacercare, sostituiscicon[
, inizio[, conteggio[, tipodiconfronto]]])
```

In cui: *stringa* è la stringa di partenza, *stringadacercare* è la sottostringa da cercare e *sostituiscicon* è la sottostringa che deve essere sostituita a *stringadacercare* (se presente). Gli altri parametri opzionali:

- **inizio** specifica la posizione da cui far partire la ricerca, se viene omesso la ricerca inizia dal primo carattere.
- **conteggio** indica il numero di sostituzioni che si devono compiere, se viene omesso vengono eseguite tutte le sostituzioni possibili.
- **tipodiconfronto** indica la modalità di ricerca.

Questa funzione può essere di notevole aiuto per eliminare tutti quei caratteri che possono mandare in tilt una ricerca su database (come gli accenti) e sostituirli con un carattere jolly:

```
Testo = "Quantita'"
Testo = Replace(Testo, "'", "*")
MessageBox.Show(Testo)   'Visualizza "Quantita*"
```

GESTIONE DI DATE E ORE

Per ottenere la data e l'ora corrente sono disponibi-

Se un'espressione contiene moltiplicazioni e divisioni, ciascuna operazione viene valutata da sinistra a destra nell'ordine in cui è indicata. Lo stesso criterio vale per addizioni e sottrazioni. Per intervenire sull'ordine di precedenza e far in modo che una parte venga valutata prima di altre, è possibile utilizzare le parentesi. Le operazioni racchiuse tra parentesi vengono eseguite prima delle altre ma all'interno delle parentesi viene mantenuto il normale ordine di precedenza tra gli operatori.

NOTA

Quando si compiono delle operazioni è solitamente necessario che gli operandi siano dello stesso tipo. Se ad esempio si somma una variabile Decimal, è necessario che il secondo termine sia anch'esso di tipo Decimal e che anche la variabile cui viene assegnato il valore sia di tipo Decimal. Per garantire l'indipendenza dai tipi si può utilizzare Option Strict. Se Option Strict è On, vengono eseguite automaticamente le conversioni dei tipi. Se si cerca di sommare una variabile Integer ad una variabile Decimal e di assegnare il valore ad una variabile Decimal, l'operazione viene eseguita normalmente. Se al contrario si cerca di sommare una variabile Integer ad una variabile Decimal e di assegnare il valore ad una variabile Decimal, viene generato un errore di compilazione, perché una variabile Decimal non può essere convertita in modo implicito nel tipo Integer. Se Option Strict è Off, queste conversioni di restrizione implicite sono permesse, ma possono determinare una perdita imprevista di dati o di precisione.

li le funzioni:

- **Today.** Restituisce la data corrente di sistema.
- **Now.** Restituisce la data e l'ora correnti, in base all'impostazione dell'ora e della data di sistema.
- **TimeOfDay.** Restituisce l'ora di sistema corrente.

Per estrarre valori da una data sono disponibili diverse funzioni che restituiscono dei numeri interi:

- **Day.** Restituisce un intero compreso tra 1 e 31 che rappresenta il giorno del mese. Prestate attenzione, per utilizzare la funzione *Day*, è necessario definirla in modo completo preceduta dal namespace *Microsoft.VisualBasic* poiché la parola chiave *Day* è definita anche nel namespace *System.Windows.Forms.*

```
MessageBox.Show(Microsoft.VisualBasic.Day(
                                        "05/02/2004"))
'Visualizza 5
```

- **Month.** Restituisce un intero compreso tra 1 e 12 che rappresenta il mese.

```
MessageBox.Show(montf("05/02/2004")) 'Visualizza 2
```

- **Year.** Restituisce un intero compreso tra 1 e 9999 che rappresenta l'anno.
- **Weekday.** Restituisce un intero che rappresenta il giorno della settimana .
- **Hour.** Restituisce un intero compreso tra 0 e 23 che rappresenta l'ora del giorno.
- **Minute.** Restituisce un intero compreso tra 0 e 59 che rappresenta i minuti.
- **Second.** Restituisce un intero compreso tra 0 e 59 che rappresenta i secondi.

Un ultima funzione che può essere utilizzata per estrarre qualsiasi valore di data od ora è la funzione *DatePart.* La sintassi della funzione *DatePart* è:

```
DatePart(intervallo, data[,primogiornodellasettimana[
                                , primasettimanadellanno]])
```

In cui *intervallo* (vedi box) rappresenta l'intervallo di tempo che si deve restituire (anno, ora, ..) e *data* rappresenta la data da testare. Gli altri due argomenti sono facoltativi ed il loro valore, se non specificato, dipende dalle impostazioni di sistema, in particolare primogiornodellasettimana, se non impostato, corrisponde a *Domenica* e *primasettimanadellanno,* se non impostato, corrisponde alla settimana nella quale cade il primo Gennaio.

OPERAZIONI CON LE DATE

Per compiere operazioni aritmetiche con le date VB fornisce due funzioni: *DateAdd* e *DateDiff.* La funzione *DateAdd* consente di aggiungere ad una data un determinato valore di tempo, la sintassi è la seguente:

```
DateAdd(intervallo, numero, data)
```

In cui: *intervallo* rappresenta il tipo d'intervallo di tempo che si vuole aggiungere, *numero* indica il numero d'intervalli da aggiungere e *data* rappresenta la data e l'ora che si vuole manipolare.
I possibili valori di intervallo sono gli stessi che può assumere nella funzione *DatePart* vista in precedenza. Il valore di numero può essere un valore negativo, in questo caso tale valore sarà sottratto alla data desiderata.
Alcuni esempi: per aggiungere dieci anni al cinque febbraio 2004 si può scrivere:

```
MessageBox.Show(DateAdd("yyyy", 10,
                "05/02/2004")) 'Visualizza 05/02/2014
```

Per sottrarre tre mesi al cinque febbraio 2004 si può scrivere:

```
MessageBox.Show(DateAdd("m",
            -3, "05/02/2004"))  'Visualizza 05/11/2003
```

La funzione *DateDiff* permette di valutare l'intervallo di tempo trascorso tra due date, per cui è possibile valutare il numero di giorni che trascorreranno da quando leggerete l'articolo a quando scriverete gli esempi successivi. La sintassi è la seguente:

```
DateDiff (intervallo, data1, data2 [
   ,primogiornodellasettimana[, primasettimanadellanno]])
```

In cui data1 e data2 rappresentano le date di cui si vuole calcolare la differenza di intervallo di tempo. Per gli argomenti intervallo, primogiornodellasettimana e primasettimanadellanno valgono le considerazioni esposte in precedenza. Nell'esecuzione del calcolo viene eseguita l'operazione *data2-data1*, per questo se *data1* è maggiore di *data2* il risultato sarà negativo

```
MessageBox.Show(DateDiff("m", "01/01/2004",
                        "05/02/2004"))  'Visualizza 1
MessageBox.Show(DateDiff("d", "01/01/2004",
                       "05/02/2004"))  'Visualizza 35
MessageBox.Show(DateDiff("d", "05/02/2004",
                      "01/01/2004"))  'Visualizza -35
```

Luigi Buono

Le fondamenta del codice "robusto"

La gestione delle eccezioni

parte seconda

Questo mese scopriremo come fare per ottenere un dettagliatissimo controllo delle eccezioni che possono potenzialmente verificarsi nei nostri software.

Nel corso del precedente appuntamento abbiamo approcciato i meccanismi basilari per la gestione delle eccezioni con C# e, più in generale, con .NET. Grazie alla gestione delle eccezioni, lo ripeto, è possibile sviluppare applicazioni più robuste, cioè in grado di fronteggiare e risolvere ogni tipo di situazione inattesa. Situazioni che, per l'appunto, sono eccezioni al normale funzionamento del software. In questa lezione proseguiremo lo studio intrapreso nel corso del mese precedente, scoprendo ulteriori caratteristiche del meccanismo di gestione delle eccezioni con C#.

LANCIO MANUALE DI UN'ECCEZIONE

Finora abbiamo affrontato situazioni in cui alcune istruzioni possono lanciare e diffondere una o più eccezioni. Ad ogni modo, se le esigenze lo richiedono, anche il nostro codice può lanciare volontariamente un'eccezione, di sua iniziativa. C# fornisce la parola chiave throw:

```
throw eccezione;
```

Guardiamo un banale esempio:

```
class Test {
  public static void Main() {
    try { //...
      throw new System.Exception("Sono un'eccezione
                                  lanciata manualmente");
    } catch (System.Exception e) {
      System.Console.WriteLine("Catturata un'eccezione,
                                    il cui messaggio è:");
      System.Console.WriteLine(e.Message); }
  }
}
```

La riga

```
throw new System.Exception("Sono un'eccezione
                            lanciata manualmente");
```

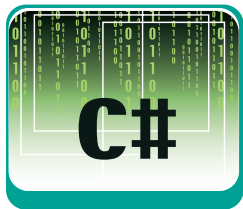
propaga un'eccezione di tipo System.Exception, il cui messaggio è "Sono un'eccezione lanciata manualmente". Siccome il comando è stato inserito all'interno di un costrutto try … catch, l'esecuzione del codice prosegue, dopo il lancio, all'interno del blocco catch, il cui compito è fornire informazioni sull'eccezione catturata.

RILANCIO DI UN'ECCEZIONE

Un blocco catch può rilanciare un'eccezione al suo esterno. La tecnica è utile quando, in presenza di più costrutti *try … catch*, si desidera fare in modo che anche i gestori catch più esterni possano interessarsi alla gestione della medesima eccezione. Il rilancio si ottiene inserendo la parola chiave *throw*, questa volta senza argomenti, al termine del blocco catch che intende effettuare il rilancio:

```
try {
  // ...
} catch (...) {
  // Codice facoltativo per la gestione dell'eccezione.
  throw; // RILANCIO!
}
```

L'utilità del rilancio viene intesa meglio esaminando un esempio concreto:

```
class Test {
  public static int dividi(int a, int b) {
    try {
```

```
      return a / b;
    } catch (System.DivideByZeroException e) {
      System.Console.WriteLine("Tentata una divisione
                                          per zero!");
      throw; // Eseguo il rilancio.}
  }
  public static void Main() {
    try {
      System.Console.WriteLine(dividi(3, 0));
    } catch (System.DivideByZeroException e) {
      System.Console.WriteLine("Divisione per zero
                                rilanciata da dividi()");
    }
  }
}
```

```
F:\c#>csc With.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

F:\c#>With
Error occurred, unable to compute
Attempted to divide by zero.

F:\c#>
```

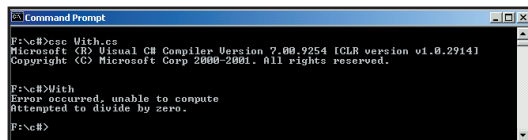



Fig. 1: *Il messaggio di errore che si ottiene se non viene gestita la tipica eccezione "Divide by zero".*

Il metodo statico *dividi()* esegue una comune divisione tra interi. Come già sappiamo, questa operazione è rischiosa, poiché il tentativo di divisione per zero causa il lancio di una *DivideByZeroException*. Il metodo *dividi()* gestisce questa eventualità, ed il blocco catch incaricato stampa in output il messaggio "Tentata divisione per zero". Dopo aver stampato l'avvertimento, il blocco catch rilancia l'ecceziona catturata. In questo modo, la stessa viene ritrasferita al codice chiamante, cioé al blocco *try* del metodo *Main()*. L'eccezione passa quindi in gestione al blocco catch del metodo *Main()*, che stampa un ulteriore messaggio di avvertimento: "Divisione per zero rilanciata da dividi()". In questo modo, dunque, è possibile far sì che i propri metodi siano in grado tanto di gestire un'eccezione quanto di ritrasmetterla al codice chiamante, per ottenere gestioni differenti in diversi punti del programma.

FINALLY

Quando le eccezioni vengono lanciate, nel codice si verifica un salto inatteso verso un blocco *catch*. Il blocco *try* che ha causato l'anomalia viene abbandonato, per non essere più ripreso. Talvolta, questo approccio comporta dei problemi. Ad esempio, se si apre un canale verso una risorsa esterna, si desidera sempre che il canale venga chiuso quando il suo utilizzo non è più necessario. Una dimostrazione è fornita dal classico esempio della lettura di un file di testo:

```
try {
  // Punto 1: apri un canale verso il file di testo.
  // Punto 2: leggi il file di testo.
  // Punto 3: chiudi il canale.
} catch (Eccezione e) {
  // Gestisci l'eccezione.
}
```

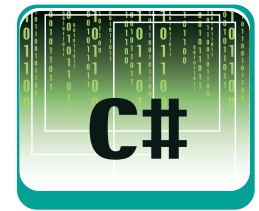
Se il punto 2 causa un'anomalia, il controllo passa al blocco catch. Il punto 3 non sarà mai eseguito, ed il canale di comunicazione resterà aperto più a lungo del dovuto. C# risolve il problema servendosi dei blocchi *finally*, che si impiegano al seguente modo:

```
try {
  // ...
} catch (Eccezione1 e) {
  // ...
} catch (Eccezione2 e) {
  // ...
} catch (Eccezione3 e) {
  // ...
} finally {
  // Codice finale
}
```

Il contenuto di un blocco *finally* viene sempre eseguito, immediatamente prima che l'intera struttura *try … catch* venga abbandonata, sia nel caso sia stata gestita un'eccezione sia in caso contrario.
La lettura di un file di testo, quindi, è più efficiente se resa alla seguente maniera:

```
// Dichiara un riferimento per il canale di comunicazione.
try {
  // Apri il canale verso il file di testo.
  // Leggi il file di testo.
} catch (Eccezione e) {
  // Gestisci l'eccezione.
} finally {
  // Chiudi il canale.
}
```

Un blocco *finally* viene sempre e comunque eseguito, anche nel caso in cui dentro il blocco try o dentro i blocchi catch vi siano istruzioni di salto che apparentemente dovrebbero impedirne l'esecuzione. Ecco un esempio:

```
class Test {
  public static void test() {
    try {
      System.Console.WriteLine("Dentro try");
      throw new System.Exception();
    } catch (System.Exception e) {
      System.Console.WriteLine("Dentro catch");
      return;
    } finally {
      System.Console.WriteLine("Dentro finally");
    }
```

```
  }
  public static void Main() {
    test();
  }
}
```

Il metodo *test()* contiene una struttura *try … catch*. Dentro il blocco *try*, intenzionalmente, viene propagata un'eccezione, cosicché il controllo passi al blocco catch. Al suo interno viene comandato un salto, che porta al termine dell'esecuzione del metodo *test()*. Nonostante questo, il contenuto del blocco finally viene ugualmente valutato, subito prima che il salto comandato abbia effettivamente luogo. Infatti, l'output riscontrabile è:

Dentro try
Dentro catch
Dentro finally

Lo stesso accade quando il salto viene comandato dall'interno del blocco *try*, come nel seguente caso:

```
class Test {
  public static void test() {
    try {
      System.Console.WriteLine("Dentro try");
      return;
    } catch (System.Exception e) {
      System.Console.WriteLine("Dentro catch");
    } finally {
      System.Console.WriteLine("Dentro finally");
    }
  }
  public static void Main() {
     test();
  }
}
```

Qui non si cade mai nel blocco *catch*. In compenso, è direttamente il blocco *try* che lancia il salto *return*. Il blocco *finally* viene eseguito ugualmente, prima che il salto abbia luogo:

Dentro try
Dentro finally

Lo stesso accade con i blocchi try annidati l'uno dentro l'altro:

```
class Test {
  public static void test() {
    try {
      System.Console.WriteLine("Dentro try esterno");
      try {
        System.Console.WriteLine("Dentro try interno");
        throw new System.Exception();
      } catch (System.DivideByZeroException e) {
        System.Console.WriteLine("Dentro catch interno");
      } finally {
        System.Console.WriteLine("Dentro finally interno");}
    } catch (System.Exception e) {
      System.Console.WriteLine("Dentro catch esterno");
    } finally {
      System.Console.WriteLine("Dentro finally esterno");}
  }
  public static void Main() { test(); }
}
```

Il blocco try più interno lancia un'eccezione che il blocco catch corrispondente non sa gestire. Quindi, il controllo del flusso passa al catch esterno. Di conseguenza, l'intera struttura *try … catch* più interna viene definitivamente abbandonata, così come viene abbandonata l'esecuzione del blocco *try* più esterno. Prima che ciò accada, viene eseguito il blocco finally della struttura interna:

Dentro try esterno
Dentro try interno
Dentro finally interno
Dentro catch esterno
Dentro finally esterno

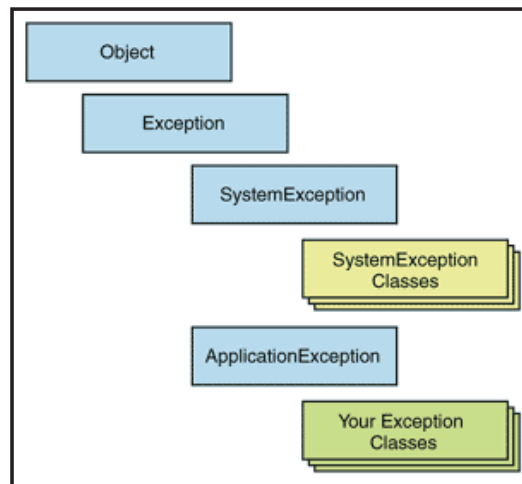



Fig. 2: ***Eccezioni: gerarchia delle classi.***

CONCLUSIONI

Con la lezione odierna abbiamo approfondito l'impiego delle clausole try … catch, imparando come avere il completo controllo delle eccezioni che percorrono il nostro software. Con la prossima lezione chiuderemo l'argomento, andando a scavare nei meandri della classe *System.Exception* e apprendendo come sia possibile realizzare eccezioni personalizzate da impiegare nelle proprie librerie e nei propri programmi. Vi aspetto dunque per il terzo ed ultimo appuntamento di questo corso dedicato alla gestione delle eccezioni.

Carlo Pelliccia

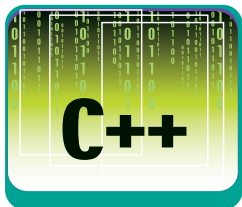
• GUIDA A C#
Herbert Schildt
(MCGRAW-HILL)
ISBN 88-386-4264-8
2002

• INTRODUZIONE A C#
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• C# GUIDA PER LO SVILUPPATORE
***Simon Robinson* e altri**
(Hoepli)
ISBN 88-203-2962-X
2001

Ordinamento: algoritmi pronti all'uso

Gli algoritmi in STL

Dopo aver parlato dei contenitori e degli iteratori, diamo un'occhiata agli algoritmi che la STL mette a nostra disposizione, che possiamo considerare come la base per l'implementazione di buona parte del nostro codice.

Nelle due precedenti puntate abbiamo visto cosa sono i contenitori (cioè, un modo per raggruppare oggetti) e cosa sono gli iteratori (una astrazione di un metodo di accesso ad un contenitore). Mediante contenitori ed iteratori possiamo già guadagnare una certa robustezza e un certo ordine nei nostri programmi, tuttavia essi sono stati creati principalmente per operare con gli algoritmi che la STL fornisce: contenitori ed iteratori non sono molto utili se poi per ogni operazione "banale" dobbiamo reinventare la ruota. In effetti ci sono delle operazioni che, data la frequenza con cui vengono eseguite, dovrebbero essere sempre disponibili, ad esempio la ricerca di un oggetto in un contenitore, oppure l'ordinamento degli elementi secondo un certo criterio. Per tutte queste operazioni, che si suppone vengano utilizzate spesso quando si lavora coi contenitori, è presente, nella libreria STL, (almeno) una implementazione sotto forma di algoritmo. Tutte le volte che abbiamo bisogno di particolari operazioni sui contenitori, conviene sempre controllare che tali operazioni non siano già implementate mediante qualcuno degli algoritmi standard.

Gli algoritmi presenti nella STL sono estremamente generici, sono cioè utilizzabili, sotto opportune ipotesi, praticamente con ogni tipo di contenitore (anche definito da noi): questo perché ogni algoritmo è definito mediante template (e in effetti STL sta proprio, come abbiamo visto, per *Standard Template Library*). Per utilizzare gli algoritmi forniti dalla STL (e disponibili nel namespace standard *std*) dobbiamo premurarci di inserire la direttiva:

```
#include <algorithm>
```

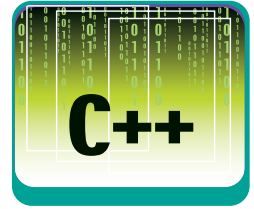
L'uso degli algoritmi forniti nel namespace standard consente di ottenere un altro importante vantaggio: spesso gli algoritmi sono semplici porzioni di codice, che possono essere inserite *inline* invece che come chiamate a funzione, ottenendo un codice compilato molto più efficiente (anche se di dimensioni maggiori). L'inserimento di codice inline è un "suggerimento" che diamo al compilatore dicendogli: *"Replica il codice della seguente funzione ogni volta che essa è invocata da qualche parte nel programma, anziché inserire una normale chiamata a funzione"*. Questo consente di aumentare la velocità di esecuzione in quanto si evitano operazioni costose come l'inserimento della funzione nello stack di attivazione o il salto a una parte lontana di memoria.

In ogni caso, gli algoritmi della STL sono realizzati in modo da avere sempre l'implementazione teorica più efficiente possibile.

I FUNCTION OBJECTS

Connesso strettamente agli algoritmi della STL è il concetto di oggetto funzione (o *function object*). È tramite i function object che possiamo far entrare il nostro codice personalizzato all'interno degli algoritmi predefiniti della STL. Questo ci consente di usare tali algoritmi anche coi nostri contenitori, che sono ovviamente diversi da quelli predefiniti. Dal punto di vista del codice, i function objects sono degli oggetti per i quali è stato ridefinito (mediante overloading) il comportamento dell'operatore *operator()()*, cioè le parentesi tonde: in questo modo, questi oggetti possono essere usati alla stregua di funzioni (e da questo deriva infatti il loro nome). Vediamo un semplice esempio:

```
#include <iostream>
using namespace std;

class FunzioneDoppio
{
  public:
     void operator()(int &x)
```

```
    {
        x = 2*x;
    }
};

int main()
{
    int x = 1;
    FunzioneDoppio f;

    cout << "X vale " << x << endl;
    f(x);
    cout << "Adesso X vale " << x << endl;

    return 0;
}
```

Come si può vedere nel corpo della funzione *main()*, abbiamo usato un oggetto *FunzioneDoppio* come fosse una funzione (e non solo nel senso della sintassi).
Creando i nostri oggetti funzione, li potremo poi usare in tutti quegli algoritmi dove questo è possibile.
Ad esempio, tra gli algoritmi a nostra disposizione c'è *for_each*, il quale ha la seguente firma:

```
template<class In, class Op>
Op for_each(In first, In last, Op f)
```

In questa firma, In rappresenta un iteratore di tipo *Input* (cioè, come abbiamo visto nella scorsa puntata, un iteratore che consente solo operazioni di estrazione di valori dagli elementi cui si riferisce), mentre *Op* è un generico tipo che abbiamo usato nella definizione del template il cui significato sarà a breve chiaro.
La funzione che questo algoritmo assolve è semplice ma allo stesso tempo estremamente utile: prende l'intervallo specificato dagli iteratori agli estremi *first* e *last* e applica a tutti gli elementi compresi in tale intervallo la funzione specificata da *f.* Si noti la potenza che ha questo algoritmo, grazie al fatto di essere definito mediante template: non importa cosa siano gli oggetti della sequenza che utilizzeremo, e non importa nemmeno cosa sia l'oggetto funzione che invocheremo; questo comporta che esso potrà essere usato con i nostri contenitori oppure con quelli predefiniti nella libreria standard.
Facciamo un esempio, usando il contenitore *vector* della STL:

```
vector<int> numeri;
for(j=0;j<numeri.size();j++)
            //...inseriamo un numero...

FunzioneDoppio f;

for_each(numeri.begin(),numeri.end(),f);
```

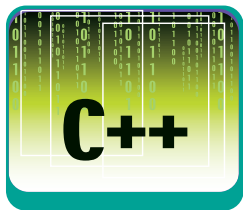
In questo modo, riempiamo dapprima il nostro contenitore di interi, quindi raddoppiamo tali interi uno ad uno.
Esistono un certo numero di tipi di function object predefiniti nella libreria STL, che sono classificati così come vengono classificati gli iteratori; il criterio di classificazione è dato dal numero di argomenti che l'oggetto prende in considerazione, dal tipo del risultato restituito (se è o no un booleano), e dalle funzionalità eventualmente offerte.
Per fare alcuni esempi, tra di essi troviamo:

- **Generator:** come il nome suggerisce, l'oggetto funzione non ha argomenti, e restituisce un oggetto di un generico tipo;
- **UnaryFunction:** l'oggetto funzione ha un argomento;
- **BinaryFunction:** l'oggetto funzione ha due argomenti;
- **Predicate:** l'oggetto funzione restituisce un valore booleano a partire dal suo (unico) argomento;
- **BinaryPredicate:** l'oggetto funzione restituisce un valore booleano a partire dai suoi due argomenti.

Oltre questi, esistono altri tipi predefiniti di oggetti funzione che possono comunque tornare utili nei nostri programmi, e che aiutano molto in caso si voglia scrivere programmi con codice il più possibile generico (e quindi portabile). Per usarli, basta includere l'header *functional.h*.

CLASSIFICAZIONE DEGLI ALGORITMI

Gli algoritmi definiti nella STL (che sono circa 60) possono essere classificati in diversi gruppi, a seconda delle principali caratteristiche che presentano:

- *Non Modifying Sequence;*
- *Modifying Sequence;*
- *Sorted Sequences;*
- *Set Algorithms;*

- *Heap Operations;*
- *Minimum and Maximum;*
- *Permutation.*

Gli algoritmi che appartengono alla classe *"Non Modifiyng Sequence"* sono quelli la cui applicazione alla sequenza di oggetti non modifica la sequenza stessa (si noti: non si possono modificare nemmeno gli elementi della sequenza!). Analogamente, gli algoritmi che appartengono alla classe *"Modifying Sequence"* modificano (come il nome stesso suggerisce) la sequenza che viene loro passata.

APPROFONDIMENTI

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro THINKING IN C++ - 2ND ED. - VOLUME 2" *Bruce Eckel* e *Chuck Allison* che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html.

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo: http://www.cplusplus.com/ref/cstring/

Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL: http://www.icce.rug.nl/documents/ che merita di essere visto (e letto) almeno una volta.

Gli algoritmi che si occupano dell'ordinamento e della gestione di sequenze di elementi, e della ricerca in sequenze ordinate di elementi, appartengono alla classe *"Sorted Sequence"*, mentre tutti quegli algoritmi che gestiscono insiemi di elementi appartengono alla classe *"Set Algorithms"* (eseguono ad esempio operazioni quali la intersezione o differenza tra insiemi di elementi). C'è poi la classe di algoritmi che si occupa di gestire un insieme di elementi come fosse un heap, e che è la classe *"Heap Operations"*. Gli algoritmi che si occupano di selezionare il minimo o il massimo tra un gruppo di elementi appartengono alla classe *"Minimum and Maximum"*. Alla classe *"Permutation"* appartengono invece gli algoritmi che si occupano di generare diverse permutazioni di un insieme di elementi.
L'algoritmo *for_each* che abbiamo citato negli esempi del paragrafo precedente, appartiene alla classe di algoritmi che non modificano la sequenza su cui vengono chiamati a lavorare: si noti però una particolarità, e cioè che questo è l'unico algoritmo di questa classe che può modificare gli elementi della sequenza (ma non la sequenza). Ad esempio, l'algoritmo *unique_ copy* prende un gruppo di elementi e vi elimina i duplicati: questo ovviamente è un algoritmo che modifica la sequenza su cui è chiamato. Un esempio di algoritmo della classe *"Sorted Sequence"* è proprio il *sort*, che si occupa di ordinare una sequenza; una volta ordinata tale sequenza ad esempio si potrebbe effettuare una ricerca su di essa usando l'algoritmo *binary_ search*, che implementa il metodo di ricerca binaria (molto efficiente per sequenze ordinate di elementi). Importante è notare che ogni algoritmo ha dei vincoli sugli iteratori che devono essere rispettati affinchè esso possa essere applicato ad una sequenza. Ad esempio, l'algoritmo *sort* richiede un iteratore di tipo *Random Access*, mentre gli algoritmi della classe *"Non Modifying Sequence"* richiedono iteratori di tipo *Input*. Oltre a questi algoritmi, ce ne sono altri di tipo più specificamente numerico, accessibili mediante l'inclusione dell'header *numeric.h*.

ORDINAMENTO

Vediamo ora qualche esempio pratico di applicazione di un algoritmo della STL. Uno di quelli utilizzati maggiormente è senza dubbio l'algoritmo di ordinamento (*sorting* in inglese). Questo algoritmo prende, come dato di ingresso, un contenitore e restituisce in uscita lo stesso contenitore, modificandolo in modo che i suoi dati siano ordinati secondo un certo criterio (crescente, decrescente, parziale ecc.). Si badi che, nella maggioranza dei casi, il contenitore restituito è lo stesso contenitore di ingresso modificato e non una sua copia, per cui, se si necessita di avere entrambi i contenitori (quello originale e quello ordinato) si dovrà provvedere a realizzare una copia manuale prima di applicare l'algoritmo.
L'estrema flessibilità di questa implementazione deriva dal fatto che il metodo di comparazione tra due oggetti (utilizzato per effettuare i vari confronti che permettono l'ordinamento) può essere deciso dallo sviluppatore. Si potrà scegliere se privilegiare efficienza di esecuzione o semplicità di scrittura del codice.
Gli algoritmi di sorting della STL si dividono in due grandi categorie: quelli che utilizzano l'operatore di comparazione *operator<()* proprio della classe cui appartengono gli elementi del contenitore, e quelli che si affidano a un *function object*, utilizzando *operator()(oggetto1, oggetto2)*.
Supponiamo ad esempio di volere ordinare un vector di interi:

```
//riempio il vettore con numeri casuali non ordinati
vector<int> vint;
```

```
vint.push_back(10);
vint.push_back(1);
vint.push_back(-3);
vint.push_back(4);
vint.push_back(0);

//ordino il vettore
sort(vint.begin(),vint.end());

//stampo il vettore ordinato a schermo
for (int i=0;i<vint.size();i++)
   cout << vint[i] << endl; //stampa: -3 0 1 4 10
```

Niente di più facile dunque! La funzione *sort()* prende come argomenti i riferimenti alle posizioni iniziale e finale del vettore, richiamati in questo caso tramite le funzioni *begin()* e *end()* di *vint*. In questo caso abbiamo ordinato un vettore di *int*. Quindi, per la comparazione tra elementi, è stato utilizzato il comune operatore < matematico. Se avessimo voluto invece ordinare un vettore di oggetti definiti da noi (ad esempio il nostro caro oggetto *"Scheda"*), avremmo potuto tranquillamente utilizzare la funzione *sort()* in modo assolutamente identico. L'unica cosa che avremmo dovuto preoccuparci di fare sarebbe stata ridefinire in maniera appropriata l'operatore *operator<()* in modo da stabilire un criterio di comparazione tra due oggetti *Scheda* (ad esempio un ordinamento alfabetico sul campo *"Nome"*).
Un metodo alternativo a questa ridefinizione è la creazione di un apposito function object che effettui la comparazione. Questo metodo è di solito utilizzato quando l'operatore < non è adatto allo scopo. Questo può avvenire, ad esempio, quando l'operatore non è proprio definito, oppure quando il suo utilizzo risulta poco efficiente. Supponendo quindi di avere definito tale oggetto (chiamiamolo *ComparatoreScheda*), potremmo ordinare un vettore di schede nel seguente modo:

```
vector<Scheda> vscheda;
//... riempio qui il vettore ...

//istanzio il function object
ComparatoreScheda cs;

//ordino il vettore
sort(vscheda.begin(),vscheda.end(),cs);
```

La chiamata a *sort()* è quindi identica, tranne che per l'utilizzo di *ComparatoreScheda* come terzo argomento della lista parametri. La STL mette a disposizione diversi tipi di algoritmi di ordinamento, ognuno dei quali potrebbe dimostrarsi più efficiente in una particolare situazione; diamo un occhiata a quelli più utilizzati:

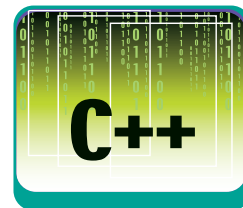
- **sort()**: è quello che abbiamo appena utilizzato. Fornisce l'ordinamento di un contenitore al quale è possibile accedere mediante iteratori di tipo *RandomAccessIterator* (cioè ad accesso casuale).
  Da notare che se il contenitore che utilizziamo non ha questo tipo di iteratore la *sort()* non è applicabile. Ad esempio per il contenitore *list* è presente una apposita *sort()* utilizzabile solo con oggetti di questo tipo.

- **stable_sort()**: questo algoritmo è analogo a *sort()* con la differenza che preserva l'ordinamento originale di elementi ritenuti equivalenti. Se ad esempio implementiamo il nostro function object per la classe *Scheda* in modo tale da ritenere equivalenti schede che presentino lo stesso campo *"Nome"*, l'ordine col quale compaiono gli altri campi ("Telefono" ad esempio) resterà quello originale.

- **partial_sort()**: si tratta di un ordinamento parziale del contenuto di un contenitore. La funzione accetta tre iteratori anziché due. L'iteratore aggiuntivo è il *"middle"*, cioè la posizione di mezzo fino alla quale si assicura l'ordinamento del contenitore, tutti gli elementi oltre questa posizione non hanno un ordinamento garantito. Questa funzione è utile, ad esempio, se si vogliono sapere le prime n posizioni di un dato contenitore, ma non ci interessa ordinarlo tutto: in questo caso possiamo risparmiare tempo di elaborazione prezioso.

CONCLUSIONI

Gli algoritmi standard, come si è visto, sono estremamente potenti e flessibili e rappresentano forse il motivo stesso dell'esistenza della STL. La loro conoscenza da parte del programmatore C++ è secondo noi un requisito fondamentale, per cui ci sentiamo di consigliare l'acquisto di un buon libro dedicato esclusivamente a questo argomento (oltre che l'assidua lettura di queste pagine! :-).
Nella prossima puntata continueremo il nostro viaggio attraverso gli algoritmi standard per dare un quadro d'insieme più completo.
Non mancate!

Alfredo Marroccelli
Marco Del Gobbo

Se avete suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Alla scoperta dei metodi

Semplice è bello

Eccoci di ritorno, pronti a rimettere mano alle nostre amate classi Incrocio e Semaforo. Non li sopporti più? Pazienta ancora per un po': il Semaforo non ha ancora finito di mostrarti quello che sa fare.

NOTA

OBIETTIVI DI QUESTA LEZIONE

- Diventerai più bravo a rimuovere le duplicazioni dal tuo codice.
- Imparerai la differenza tra interfaccia e implementazione.
- Farai la conoscenza della parola chiave private.

NOTA

LO DICO UNA VOLTA SOLA

Devi sempre fare tutto il possibile per evitare le duplicazioni nel tuo codice. Molti esperti di software ritengono che il programma ideale sia quello che dice ciascuna cosa una ed una sola volta ("once and only once"), senza mai ripetersi.

Anche questa volta cominceremo dando un'occhiata al codice del mese scorso, che puoi trovare anche su *www.ioprogrammo.it*. Come al solito ti consiglio di leggere questo articolo davanti al computer, per sperimentare direttamente quello che leggerai.
Torniamo alle solite classi *Incrocio* e *Semaforo*. Il mese scorso avevamo spostato buona parte del codice nel *Semaforo*. Il *Semaforo* è diventato più intelligente e di conseguenza l'*Incrocio* è diventato più semplice:

```
class Incrocio {
 public static void main(String[] args){
  Semaforo s = new Semaforo();
  for(int i = 1; i <= 5; i++) {
   // il Semaforo diventa verde
   s.verde = true;
   s.giallo = false;
   s.rosso = false;
   s.stampaStato();
   s.stampaComando();
   // il Semaforo diventa giallo
   s.verde = true;
   s.giallo = true;
   s.rosso = false;
   s.stampaStato();
   s.stampaComando();
   // il Semaforo diventa rosso
   s.verde = false;
   s.giallo = false;
   s.rosso = true;
   s.stampaStato();
   s.stampaComando(); }
 } }
class Semaforo {
    boolean verde = true;
    boolean giallo = false;
    boolean rosso = false;
    boolean go() {
      if(rosso || giallo)
        return false;
      return true; }
    void stampaComando() {
      if(go())
        System.out.println("Go!");
      else
        System.out.println("Stop!");}
    void stampaStato() {
      System.out.println("V: " + verde + ", G: " + giallo
                                    + ", R: " + rosso); }
}
```

Ed ecco il diagramma del *Semaforo* in notazione UML:

Semaforo	
verde:	*boolean*
giallo:	*boolean*
rosso:	*boolean*
go():	*boolean*
stampaComando():	*void*
stampaStato():	*void*

Il mese scorso ti ho lasciato con una sfida abbastanza difficile: ti ho chiesto di eliminare tutte le duplicazioni di codice nella classe *Incrocio*. Proviamo a risolvere questo esercizio passo per passo.

LA RADICE DI OGNI MALE

Ora l'*Incrocio* delega buona parte delle sue responsabilità al *Semaforo*. È il *Semaforo* che stampa il proprio stato, ed è sempre esso che stampa il comando per gli automobilisti. Mi sembra una scelta piuttosto naturale: se si pensa al mondo reale viene spontaneo immaginare un *Semaforo* che si occupa personalmente di queste faccende, mentre il suo "client" (in questo caso l'*Incrocio*) si limita a dargli qualche semplice ordine. Il vantaggio di questo approccio, come forse ricordi dagli esercizi del mese scorso, è che l'*Incrocio* non è costretto a sapere come funziona il *Semaforo:* gli basta sapere quali ordini gli può dare, cioè quali sono i suoi metodi. Purtroppo abbiamo ancora un sacco di duplicazioni nella classe *Incrocio.* Le operazioni di stampa sono ripetute tre volte, esattamente identiche.
Anche il codice che cambia lo stato delle luci del Semaforo è piuttosto antipatico, per due motivi.

Primo: è ripetuto tre volte - non proprio identico, ma per lo meno molto simile. Abbastanza simile, direi, da puzzare di codice duplicato (che come avrai ormai capito è alla radice di tutto il male nell'universo).

Secondo: per colpa di questo codice l'*Incrocio* è costretto a sapere troppe cose del *Semaforo,* e ad assumersi responsabilità che in realtà non gli spettano. Ad esempio l'*Incrocio* deve sapere che il *Semaforo* ha tre luci rappresentate da tre campi booleani. Deve anche decidere come funzionano queste luci: è l'*Incrocio* che impone ad esempio la regola per la quale le luci rossa e gialla non possono mai essere accese contemporaneamente. Cosa succederebbe se volessimo cambiare il numero delle luci del *Semaforo,* o le regole secondo le quali si accendono e si spengono le luci? Dovremmo per forza modificare l'*Incrocio*. Invece al nostro *Incrocio* piacerebbe solo ordinare al *Semaforo*: "scatta!", e dovrebbe essere il *Semaforo* stesso ad accendere e spegnere le luci giuste. Proviamo a trasferire nel *Semaforo* il codice che gestisce le luci. Non solo il *Semaforo* deve ricordare la configurazione delle luci in ogni momento (cosa che fa già grazie ai suoi tre campi), ma deve anche sapere quale nuova configurazione raggiungere ogni volta che il suo client gli ordina di scattare. Per gestire questi cambiamenti possiamo fare ricorso ad una macchina a stati. Una macchina a stati è un oggetto che può trovarsi in un numero definito di stati, proprio come il nostro *Semaforo* può trovarsi negli stati "rosso", "giallo" o "verde". La macchina cambia stato quando si verifica qualche evento esterno. Il percorso attraverso gli stati è definito in base all'evento esterno.

Confuso? Niente paura: il nostro *Semaforo* è una macchina a stati molto semplice. Esiste solo un evento esterno, che è il comando per far scattare il *Semaforo.* A questo scopo aggiungeremo un metodo *scatta()* al *Semaforo.* Ogni volta che un client chiama questo metodo, il *Semaforo* deve cambiare stato. Il diagramma della macchina a stati è molto semplice, e lo puoi vedere in Fig. 1. Ciascuna freccia indica una transizione di stato. L'etichetta sulla freccia è l'evento che deve verificarsi perché avvenga la transizione

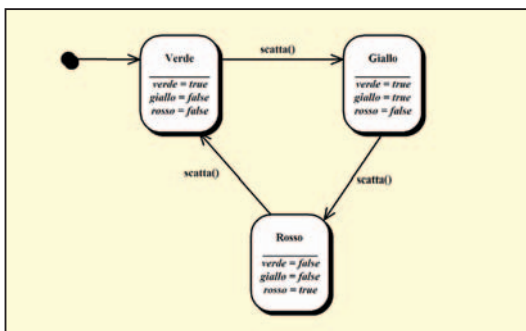



***Fig. 1:* Diagramma della macchina a stati.**

(in questo caso, la chiamata di un client al metodo *scatta()*). Il pallino nero indica lo stato iniziale del sistema. Il diagramma mostra che il *Semaforo* nasce in stato "verde" e, in seguito alle chiamate del metodo *scatta()*, passa prima in stato "giallo", poi in stato "rosso", e poi torna stato "verde" e ricomincia il suo ciclo. Il *Semaforo* attraversa questi stati per sempre, quindi in questo caso non esiste uno "stato finale".

LA MACCHINA A STATI

Eccoti una possibile soluzione all'*Esercizio 1*(se gli operatori logici ti confondono, vai a dare una ripassata ai primi articoli di questo corso):

```
class Semaforo {
 [...]
 void scatta() {
  if(verde && !giallo) {
   verde = true;
   giallo = true;
   rosso = false; }
  else if(verde && giallo) {
   verde = false;
   giallo = false;
   rosso = true; }
  else if(rosso) {
   verde = true;
   giallo = false;
   rosso = false; }
  return; }
}
```

In pratica ho separato una parte della logica del *Semaforo* dal codice che la usa.
Il primo passo per risolvere *Esercizio 2* è ovvio: anziché manipolare "a mano" il valore dei campi del *Semaforo*, l'*Incrocio* può chiamare semplicemente il metodo *Semaforo.scatta():*

```
class Incrocio {
  public static void main(String[] args){
    //crea il Semaforo
    Semaforo s = new Semaforo();
    // ripeti il ciclo del Semaforo cinque volte
    for(int i = 1; i <= 5; i++) {
      s.scatta();
      s.stampaStato();
      s.stampaComando();
      s.scatta();
      s.stampaStato();
      s.stampaComando();
      s.scatta();
      s.stampaStato();
      s.stampaComando();
    }
```

ESERCIZIO 1

Prova a scrivere da solo il metodo *Semaforo.scatta().* Questo metodo deve leggere l'attuale stato del Semaforo e passare allo stato successivo. Se il Semaforo è verde allora deve diventare rosso, se è rosso deve diventare giallo e se è giallo deve diventare verde.

ESERCIZIO 2

La nuova versione del Semaforo, sempre più intelligente, ti permette di semplificare ancora la classe Incrocio. Elimina tutte le duplicazioni. Quante righe di codice ti servono per fare le stesse cose che facevi prima?

NOTA

INTERFACCIA E IMPLEMENTAZIONE

In questo articolo distinguo l'aspetto esterno di una classe, quello che i suoi client vedono e usano, dal codice contenuto al suo interno. L'insieme dei metodi e dei campi visibili ai client si chiama interfaccia. L'interfaccia del Semaforo è quella che vedi nei diagrammi UML sparsi per l'articolo. Il funzionamento interno della classe si chiama invece implementazione. Questo mese imparerai a rendere "privati" i campi, e così facendo li "sposterai" dall'interfaccia all'implementazione.

```
  }
}
```

Ora l'*Incrocio* si è ridotto ad un semplice ciclo da 1 a 5, che al suo interno contiene lo stesso identico codice ripetuto per tre volte. Finalmente possiamo eliminare questa fastidiosa ridondanza:

```
class Incrocio {
 public static void main(String[] args){
  Semaforo s = new Semaforo();
  for(int i = 0; i <= 15; i++) {
   s.stampaStato();
   s.stampaComando();
   s.scatta(); } }
}
```

Nota che ora il ciclo va da 1 a 15, perché prima ciascuna iterazione ripeteva le stesse operazioni tre volte.

RITORNO ALL'ANAGRAFE

Prima di andare avanti, ti propongo di cambiare i nomi dei campi della classe *Semaforo*. I nomi "verde", "rosso" e "giallo", che un paio di mesi fa mi sembravano buoni, ora mi sembrano piuttosto ambigui. Ciascuna di queste tre parole può indicare due cose ben diverse: la corrispondente luce colorata, oppure, uno dei tre possibili stati del *Semaforo*. In un *Semaforo* in stato "giallo", ad esempio, il campo verde e il campo giallo valgono *true* e il campo rosso vale *false*. Per risolvere questa confusione ti propongo di rinominare i tre campi perché sia chiaro che rappresentano gli stati delle singole luci, non gli stati del *Semaforo* nel suo complesso:

```
class Semaforo {
 boolean luceVerde = true;
 boolean luceGialla = false;
 boolean luceRossa = false;
 [...]
}
```

Ecco il nuovo diagramma UML del *Semaforo*:

Semaforo	
luceVerde:	*boolean*
luceGialla:	*boolean*
luceRossa:	*boolean*
scatta():	*void*
go():	*boolean*
stampaComando():	*void*
stampaStato():	*void*

Naturalmente dovremo cambiare i nomi dei campi anche nel metodo *scatta()* e in tutti gli altri metodi del *Semaforo*. Ti ricordo che l'Incrocio non usa più questi campi, quindi puoi ricompilare la classe Semaforo senza ricompilare *Incrocio*. Prova. Quando ti sei convinto, cerca di risolvere l'*Esercizio 3*. Se scrivi il *SemaforoADueLuci* partendo dal *Semaforo* che abbiamo usato finora, potresti avere la tentazione di usare due campi booleani per rappresentare le luci rossa e verde. Pensaci bene, prima di andare avanti. Prima di darti la soluzione, però, facciamo una breve ma importante digressione.

UN PICCOLO SEGRETO

Ora il Semaforo somiglia ad un vero Semaforo: da fuori abbiamo solo dei meccanismi estremamente semplici (il metodo *scatta()*, il metodo *go()*), mentre la "roba" più complessa è nascosta al suo interno. Ad esempio, i tre campi che controllano le luci e la loro gestione non sono più necessari per i client del *Semaforo*. Quando dico che i tre campi sono nascosti, però, dico una piccola bugia. In effetti i campi sono sempre lì. Sono i client che fanno "finta" di non vederli. I client hanno tutto quello che gli serve per controllare il *Semaforo* senza toccare questi campi, ma se volessero potrebbero comunque leggerli e magari cambiarne il valore. Il *Semaforo* è come un'automobile con il cofano spalancato: chiunque può metterci le mani dentro e modificarla.
Questa non è una bella cosa, per tre motivi.

Primo: complessità inutile. Un giorno qualcuno scriverà un nuovo client che usa il Semaforo, quindi dovrà capire come funziona. Questo "qualcuno" potresti essere tu stesso, quando tra poche settimane ti sarai dimenticato come è fatta questa classe. In quel momento, a scanso di errori e perdite di tempo, sarà bene che la classe sia semplice da capire e da usare. Perché una classe sia semplice, il suo aspetto "vista da fuori" deve essere semplice e chiaro (l'aspetto esterno di una classe si chiama interfaccia. Questo può sembrare un problema da poco nel caso del semplice *Semaforo,* ma quando i tuoi programmi cominceranno ad avere decine o centinaia di classi, inizierai ad apprezzare il valore della semplicità. Come puoi semplificare l'interfaccia di una classe? Tutto quello che non è indispensabile all'interfaccia andrebbe eliminato o nascosto. Purtroppo non riesco ad immaginare un Semaforo più semplice di questo, quindi a quanto pare non possiamo eliminare niente.

Secondo: modifiche difficili. Non possiamo essere sicuri che qualche client maleducato non decida di usare direttamente i campi. Un client invadente (magari scritto da te stesso in un momento di fretta) potrebbe decidere di usare il *Semaforo* "da dentro". A

COMPILA POCO, COMPILA BENE

Questo mese farai molte modifiche alla classe Semaforo. Una cosa interessante è che non sempre, dopo aver modificato Semaforo, sei costretto a ricompilare anche Incrocio. Se l'interfaccia di Semaforo non cambia, la classe Incrocio continua a funzionare come se niente fosse. Naturalmente, se modifichi l'"aspetto esterno" del Semaforo (ad esempio cancelli o rinomini un metodo) dovrai modificare il codice di Incrocio e ricompilarlo.

Scrivi un semaforo che non ha la luce gialla. Chiamalo SemaforoADueLuci. Questo semaforo avrà solo due stati: "verde" (luce rossa spenta, luce verde accesa) e "rosso" (luce rossa accesa, luce verde spenta). Quali metodi devi cambiare? È necessario ricompilare Incrocio o puoi lasciarlo così com'è? Di quanti cambi hai bisogno?

questo punto non sarebbe più così facile modificare l'interno del *Semaforo*. Se il *Semaforo* fosse "chiuso" potresti invece fare quello che ti pare (eliminare una delle luci, o cambiare il percorso della macchina a stati), pur di non toccare l'interfaccia. Se *Semaforo* resta "aperto", rischierai di dover modificare alcuni client ogni volta che modifichi l'implementazione.

Terzo: poca sicurezza. Un *Semaforo* aperto a tutti è un oggetto fragile, perché basta un errore di programmazione nei client per portarlo in uno stato "impossibile" (o, per dirla in gergo informatico, inconsistente).

```
Semaforo s = new Semaforo();
s.luceVerde = true;
s.luceRossa = true;
// luce rossa e luce verde
// accese contemporaneamente!
s.scatta(); // ...e ora cosa succedera'?
```

Ora il Semaforo è in uno stato inconsistente, che non è previsto dal metodo *scatta()*. Quindi non riuscirà mai più a cambiare stato! Come puoi risolvere questo problema? Non puoi eliminare i tre campi, perché sono essenziali per conservare lo stato del *Semaforo*. Però puoi nasconderli, grazie alla parola chiave *private:*

```
class Semaforo {
 private boolean luceVerde = true;
 private boolean luceGialla = false;
 private boolean luceRossa = false;
 [...] }
```

Quando usi la parola *private* prima della dichiarazione di un campo, dichiari che quel campo deve essere invisibile dall'esterno della classe. In pratica stai rimuovendo il campo dall'interfaccia, pur senza rimuoverlo dall'implementazione. La classe stessa può usare il campo esattamente come prima, ma i client non possono più scriverlo né leggerlo. Per chi vede il Semaforo "da fuori" il campo non esiste: la sua visibilità è ristretta all'interno della classe. Se cerchi di accedere ad un campo privato per leggerne o per cambiarne il valore, verrai fermato da un errore di compilazione. Come regola generale, cerca sempre di rendere privati tutti i campi che non devono assolutamente restare visibili. Più l'interfaccia è semplice, meglio è.
Ora l'interfaccia del Semaforo è davvero semplicissima:

Semaforo	
scatta():	*void*
go():	*boolean*
stampaComando():	*void*
stampaStato():	*void*

UN MONDO PIÙ SEMPLICE

Torniamo all'Esercizio 3. Forse hai usato due campi booleani per rappresentare le due luci del Semaforo. Questa è una soluzione accettabile, ma ridondante. Visto che le due luci hanno sempre valore opposto l'una all'altra (quando una è accesa l'altra è sempre spenta) puoi tranquillamente usare un solo campo booleano che rappresenta lo stato. Ho chiamato questo campo *go*. Quando *go* vale *true*, il *Semaforo* è verde; quando *go* vale *false*, il *Semaforo* è rosso. Il codice risultante è davvero molto semplice:

```
class SemaforoADueLuci {
 private boolean go = true;
 void scatta() {
    go = !go; }
 boolean go() { return go; }
 void stampaComando() { if(go())
   System.out.println("Go!");
  else
  System.out.println("Stop!"); }
}
```

Per brevità, ho eliminato il metodo *stampaStato();* se vuoi, prova a scriverlo da solo. Il mio preferito tra i tre metodi è il metodo *scatta()*, che si è ridotto ad una sola riga che inverte lo stato del campo booleano. Il metodo *go()* non fa altro che restituire il valore del campo. In un certo senso il campo *go* è diventato un campo "di sola lettura": un client può leggerne il valore attraverso il metodo *go()*, ma trattandosi di un campo privato nessuno può cambiarne il valore all'insaputa del *SemaforoADueLuci*. Se preferisci, usa pure due variabili per rappresentare le due luci. Si tratta di roba *private*, quindi hai la garanzia che nessun client si accorgerà di nulla (anche perché ancora non hai scritto nessun client, e questa è una garanzia ancora maggiore). Nota che, in linea di principio non c'è niente di male nell'avere un campo e un metodo che si chiamano nello stesso modo: si tratta comunque di due cose diverse, e il metodo sarà sempre distinguibile grazie alla coppia di parentesi tonde che lo segue. Ma il fatto che una "cosa" come un campo ed un'"azione" come un metodo abbiano lo stesso nome mi fa pensare che forse avrei potuto trovare dei nomi migliori per uno dei due, o per entrambi. Questo è un esercizio piuttosto "filosofico" e senza una soluzione ben definita. Pensaci un po' su *(Esercizio 4 e 5)*. Se hai difficoltà, troverai il *SemaforoADueLuci* e l'*IncrocioConDueSemafori* nell'area download di *ioProgrammo.it* di questo mese. Ma so che non andrai a sbirciare subito. Dopo tutto, hai un mese di tempo. Se hai qualche difficoltà, non esitare a scriverci in redazione.
Arrivederci tra trenta giorni!

Paolo Perrotta

ESERCIZIO 4

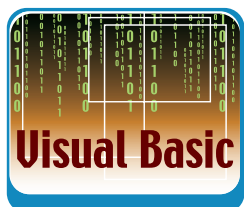
Prova a cercare dei nomi migliori per il campo *go* o per il metodo *go()*, o magari per tutti e due. Ricorda che i campi dovrebbero avere nomi di cose, e i metodi (che sono comandi) dovrebbero avere nomi di azioni. Il nome *go()* è un'azione, eppure mi sembra comunque poco adatto. Sei d'accordo? Riesci a spiegare perché?

ESERCIZIO 5

Scrivi un incrocio con due semafori. L'incrocio attraversa un certo numero di cicli (scegli tu quanti). Ad ogni iterazione, entrambi i semafori cambiano stato. Per evitare rovinosi incidenti, i due semafori devono essere "in controfase": quando uno è verde, l'altro deve essere rosso, e viceversa.

Sfruttiamo le possibilità offerte dai Web Services

Un Client per WS con MS SOAP Toolkit 3.0

Microsoft SOAP toolkit 3.0 è l'ultima release del tool della Microsoft che implementa le specifiche SOAP 1.1, con cui è possibile sviluppare client SOAP e web service.

I web service sono i nuovi "protagonisti" dell'informatica distribuita: possono essere utilizzati per vari scopi, da semplici contenitori/fornitori di informazioni, (previsioni meteorologiche, annunci di vario tipo, ecc), a veri e propri servizi standalone o integrabili in altre applicazioni. Alle funzionalità offerte dai web service si può accedere con dei client che utilizzano il protocollo SOAP *(Simple Object Access Protocol)*. Il protocollo SOAP è concepito per standardizzare lo scambio di messaggi tra computer, esso non è legato ad un particolare linguaggio di programmazione o sistema operativo. Gli elementi fondanti del SOAP sono il Remote Procedure Calls (RPC), il protocollo HTTP e l'XML. Le funzionalità dei web service (come nel caso dei più familiari componenti COM) sono offerte attraverso dei metodi. Le interfacce di questi metodi sono descritte attraverso un documento XML scritto in WSDL *(Web Services Description Language)*. Un documento WSDL è una specie di contratto tra server e client. Attualmente diversi prodotti Microsoft implementano il supporto nativo per SOAP, per esempio .NET, mentre in altri prodotti, come Visual Basic 6, si può aggiungere attraverso Microsoft SOAP Toolkit, lo strumento che introdurremo in questo articolo.
In questo appuntamento, prima introdurremo la tecnologia alla base del SOAP Toolkit e poi implementeremo due tipi di client per web service: "Client HTML" e Client SOAP.

WEB SERVICE E SOAP

Con SOAP la comunicazione avviene sul protocollo HTTP (ma può avvenire anche su SMTP, FTP, ecc) attraverso la tecnica della invocazione (incapsulata in un documento XML) di procedure remote (RPC). Un client SOAP invia un messaggio XML al server con le indicazioni sulla procedura (nome della procedura e relativi argomenti) che vuole invocare.
Il server, dopo aver eseguito la procedura invocata, invia il risultato al client sempre in un messaggio XML. Il formato dei messaggi deve rispettare lo standard SOAP mentre i nomi delle procedure ed il formato dei suoi argomenti devono essere specificati come definito nel contratto tra server e client (che è un documento WSDL). I messaggi SOAP ed i documenti WSDL, però, non devono essere scritti dallo sviluppatore del web service: come vedremo tra poco, gli strumenti e gli oggetti del Toolkit SOAP si occupano di produrre i documenti WSDL e di tradurre le chiamate a procedure, fatte in Visual Basic, in messaggi SOAP. Un messaggio SOAP è un documento XML composto da tre elementi: *Envelope, Header* e *Body*. La radice del messaggio è l'elemento *<SOAP-ENV:Envelope>* questo può contenere l'elemento *<SOAP-ENV:Header>*, che è opzionale e serve per gestire particolari caratteristiche del messaggio, e l'elemento *<SOAP-ENV:Body>* che effettivamente è il corpo del messaggio e contiene il nome della procedura da invocare ed i rispettivi argomenti (oppure il risultato prodotto dal server).

WSDL E WSML

Come accennato il WSDL è un linguaggio XML-based che serve per individuare il web service ed i servizi offerti. Per capirci, in un file WSDL, per ogni servizio che può essere invocato dal client, sono definiti la struttura delle interfacce, degli argomenti e dei valori restituiti dalle procedure. Gli sviluppatori di web service con il SOAP Toolkit possono generare il file WSDL, a partire da un oggetto COM, attraverso il Wizard del generatore di file WSDL/WSML. In realtà questo Wizard, oltre al file WSDL, crea un file WSML *(Web Services Meta Language)* che serve per associare alle operazioni descritte nel file WSDL i metodi del componente COM che implementa il servizio (il file WSML è specifico del Microsoft SOAP Toolkit).

W3C

Se volete seguire gli sviluppi dell'infrastruttura dei web service legata all'XML, sul sito del Worldwide Web Consortium (W3C), vi consigliamo i seguenti link:

http://www.w3.org/TR/SOAP
http://www.w3.org/TR/wsdl.

Questi aspetti verranno illustrati nell'appuntamento successivo. Ora possiamo vedere il primo esempio di client per web service realizzato senza utilizzare il SOAP Toolkit; infatti mostreremo come è possibile collegarsi ad un web service attraverso un file HTML. Il web service è offerto gratuitamente sul sito WebserviceX.NET

UN CLIENT HTML

Il web service che consulteremo è stato sviluppato con ASP.NET (questi web service per impostazione predefinita oltre al SOAP supportano i protocolli http-GET e http-POST). Il web service, denominato *country,* fornisce informazioni (prefisso internazionale, fuso orario ecc.) sulle principali nazioni (country) del mondo. Il file WSDL del web service può essere selezionato attraverso il seguente link: *www.webservicex.net/country.asmx?WSDL.* Facciamo notare che asmx è l'estensione dei file ASP.NET. Ecco il codice del client HTML.

```
<form method=GET
  action='http://www.webservicex.net/country.asmx/GetISD'
  name="webserver" target="_blank">
 <input type="text" size="35" name='CountryName'>
 <p> <input type=submit value="Send"> </p>
</form>
```

Con il codice precedente si crea un form HTML, con un pulsante submit denominato "Send". Notate che il form, per inviare i dati, utilizza il metodo *Get* (protocollo *http-Get*). Cliccando sul pulsante *Send,* i dati sono passati al servizio *GetISD.* Il server, dopo aver elaborato i dati risponde con un messaggio che riporta il codice telefonico internazionale del country (si veda Fig. 1). Descriveremo brevemente il Toolkit SOAP della Microsoft e poi presentiamo un esempio di client SOAP per il web service country.

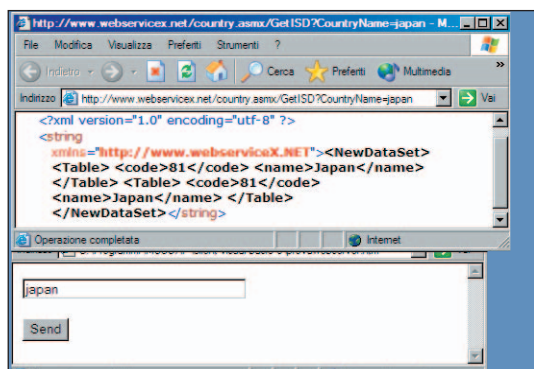



Fig. 1: *Il client HTML e la stringa XML prodotta dal web service.*

SOAP TOOLKIT

L'ultima versione del SOAP Toolkit è la 3 e può essere scaricata dal sito della Microsoft. Per installare il Toolkit su sistemi diversi da Windows XP, bisogna prima installare Windows Installer 2.0 o versioni successive (anch'esso scaricabile dal sito della Microsoft). Il Toolkit è compatibile con le raccomandazioni SOAP 1.1 e le specifiche WSDL 1.1 rilasciate dal W3C. Il SOAP Toolkit è composto dai seguenti elementi.

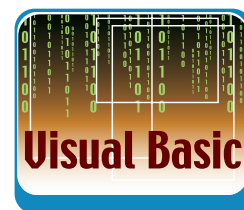
1. Il componente client che consente di invocare le funzionalità offerte dai web service;
2. Il componente server che mappa le invocazioni dei servizi in delle invocazioni di metodi COM. Questo è fatto con il supporto dei file WSDL e WSML;
3. I componenti necessari per costruire, leggere, trasmettere ed elaborare i messaggi SOAP;
4. Una dettagliata guida utente e un file *ReadMe,* quest'ultimo fornisce delle indicazioni su come risolvere i problemi d'installazione;
5. Un generatore di file *WSDL/WSML* che permette di creare i file *WSDL/WSML* a partire dai componenti COM;
6. La *Trace* utility che permette di tracciare i messaggi SOAP che il client SOAP ed il server si spediscono sull'HTTP;
7. Uno *Script* (*soapvdir.cmd*) per configurare, opportunamente, le directory virtuale dell'IIS quando si utilizza un listener ISAPI.

Il *listener* è un elemento dell'architettura del web service che gestisce le richieste SOAP. Esso può essere un file *Internet Server API* (ISAPI) oppure un file (ASP). Gli ultimi tre punti dell'elenco precedente li studieremo nel successivo appuntamento, quando vedremo l'architettura di un web service. Ora descriviamo sommariamente gli oggetti e le interfacce del SOAP Toolkit che ci consentono di implementa un client SOAP.

OGGETTI E INTERFACCE PER IL CLIENT SOAP

Nel Toolkit gli oggetti e le interfacce sono divisi in due categorie:

1. **SOAP Objects/Interfaces,** che sono gli oggetti e le interfacce (API) di alto e basso livello che permettono di creare, spedire e processare i messaggi SOAP;
2. **Modello ad oggetti per i file WSDL e WSML.**

In questo nostro primo esempio utilizzeremo alcuni elementi delle API di alto livello, quelle di basso livello, invece, le vedremo nel successivo appuntamento. In particolare utilizzeremo l'oggetto SoapClient30 che fornisce delle proprietà e dei metodi, di alto

SOFTWARE

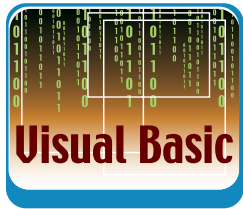
Riassumiamo le caratteristiche Software per il client ed il server che utilizzano il SOAP Toolkit 3.0. Per il client è richiesto uno dei seguenti sistemi operativi: Windows XP, Windows 2000 SP1, Windows NT 4.0 SP6, Windows 98 oppure Windows Millennium. Per il Server Soap, invece, sono richiesti Internet Server API (ISAPI) oppure Active Server Pages (ASP) sui seguenti sistemi operativi Windows XP, Windows 2000, Windows NT 4.0 SP6 e, naturalmente, Internet Information Services (IIS) 5.x (su Windows XP e Windows 2000) oppure IIS 4.0 (su Windows NT 4.0).

livello, che consentono di spedire delle richieste al server e di elaborarne le risposte. Del SoapClient30 useremo i metodo MSSoapInit, la proprietà Client-Property e le proprietà che mostrano i messaggi d'errore. MSSoapInit permette d'impostare l'oggetto SoapClient30 usando il file WSDL del web service. Dopo l'impostazione iniziale, l'oggetto SoapClient 30 può richiamare i metodi offerti dal web service. La ClientProperty, invece, la usiamo per settare la proprietà ServerHTTPRequest dato che dobbiamo caricare il file WSDL attraverso l'HTTP (facciamo una richiesta HTTP). Ora costruiamo il nostro client SOAP, attraverso un progetto Visual Basic EXE. Il client lo colleghiamo al web service Country per ricavare il fuso orario e il prefisso telefonico internazione di una determinata nazione.

Fig. 2: *La finestra dei riferimenti di Visual Basic.*

IL CLIENT SOAP

Create un nuovo progetto EXE e referenziate la libreria: *Microsoft Soap Type Library v3.0.*
Sul form inserite tre textbox *(Txtstato, Txtora, Txtcodice)*, un pulsante *(Send)* e un RichTextBox (di cui impostate le proprietà multilinea e scrollbars). Il codice da inserire in *Send_Click* è il seguente.

```
Private Sub Send_Click()
 Dim soapClient3 As MSSOAPLib30.SoapClient30
 Set soapClient3 = New MSSOAPLib30.SoapClient30
 On Error Resume Next
  soapClient3.MSSoapInit
    ("http://www.webservicex.net/country.asmx?WSDL")
  soap Client3.ClientProperty("ServerHTTPRequest") = True
  If Err <> 0 Then
   RichTextBox1.Text = "non connesso"
  Else
   RichTextBox1.Text = "connessione riuscita"
  End If
  oraxml = soapClient3.GetGMTbyCountry(Txtstato)
  pos = InStr(1, oraxml, "<GMT>")
  pos2 = InStr(1, oraxml, "</GMT>")
  ora = Mid$(oraxml, pos + 5, pos2 - (pos + 5))
  codexml = soapClient3.GetISD(Txtstato)
  pos = InStr(1, codexml, "<code>")
  pos2 = InStr(1, codexml, "</code>")
  code = Mid$(codexml, pos + 6, pos2 - (pos + 6))
```

UDDI

Per rintracciare i web service su Internet è stato creato l'UDDI (Universal Description, Discovery, and Integration) una sorta di pagine gialle dei servizi web. Sull'UDDI è possibile trovare informazioni sui Servizi, su chi li ha sviluppati, su come contattarli ecc. Si consulti www.uddi.microsoft.com

```
  txtcodice = code
  txtora = ora
  RichTextBox1.Text = RichTextBox1.Text _
  & Chr(13) + codexml + Chr(13) + oraxml
  If Err <> 0 Then
   MsgBox Err.Description
   MsgBox "faultcode=" + soapClient3.FaultCode
   MsgBox "faultstring=" + soapClient3.FaultString
   MsgBox "faultactor=" + soapClient3.FaultActor
   MsgBox "detail=" + soapClient3.Detail
  End If
End Sub
```

Quando sul form si clicca il pulsante *Send* le richieste vengono inviati al web service e nel RichTextBox vengono mostrate le stringe XML restituite. Nei due textbox – ora e codice – invece vengono mostrati i dati relativi alla Nazione (specificata in *Txtstato*). Nella procedura utilizziamo due servizi del web service: *GetISD* che abbiamo già illustrato e *GetGMTbyCountry* che restituisce il fuso orario di una nazione. Notate che dopo aver impostato l'oggetto *soapClient30* l'invocazione di una procedura remota (RPC) è simile all'invocazione di un procedura locale. Notate, inoltre, che il messaggio restituito dal server è una stringa, quindi possiamo elaborarla con le funzioni *InStr* e *Mid$* per estrarre i valori che c'interessano.

Fig. 3: *Il Form del client SOAP.*

CONCLUSIONI

In questo appuntamento abbiamo descritto, soltanto, alcune caratteristiche del Soap Toolkit ed abbiamo capito che il protocollo SOAP, sfrutta delle tecnologie per il Web già consolidate e resta indipendente dal sistema operativo, dal linguaggio di programmazione e dal protocollo di trasporto.
Nel prossimo articolo approfondiremo le conoscenze sul Toolkit e parallelamente presenteremo un esempio completo di web service. In particolare implementeremo un web service (utile alle agenzie immobiliari) che fornisce informazioni sugli appartamenti che si affittano (indirizzo, prezzo…). Per realizzarlo utilizzeremo le low level API, la "serializzazione" dei messaggi XML, un database… seguiteci!

Massimo Autiero

L'architettura basata sui Web Service

Introduzione al Service Oriented Architecture

In questo articolo vedremo come si è arrivati alla definizione di Service Oriented Architecture, in cosa consiste, e come questo tipo di architettura possa migliorare la realizzazione di applicazioni.

I sistemi informatici moderni sono progettati per essere interconnessi: le tipologie di sistemi operativi, le tecnologie software e hardware sono differenti, e nessuna ha il sopravvento sulle altre. Lo sviluppatore oggi deve partire da queste considerazioni per poter comprendere come il suo lavoro possa interfacciarsi con quello degli altri. Queste abilità permettono già ora di sfruttare tecnologie e paradigmi utili alla realizzazione di applicazioni fortemente distribuite e basate su servizi software eterogenei.
Con l'appoggio crescente da parte delle maggiori industrie del software, l'architettura basata sui servizi promette di fornire quello che CORBA (*Common Object Request Broker Architecture*) e D-COM (*Distributed COM*) non sono mai riusciti a realizzare appieno: l'utilizzo di servizi a prescindere da dove e come siano realizzati.

UNA NUOVA ARCHITETTURA

Arriviamo ad oggi, e facciamo un esempio. Poniamo di avere un problema: siamo i responsabili IT di un'azienda, e dobbiamo integrare il sistema informativo che negli anni si è venuto a creare. Sia all'interno dell'azienda che verso l'esterno sempre più si sente il bisogno di utilizzare risorse eterogenee: l'inventario è gestito su un server linux, i processi gestionali e l'application server risiedono su Windows2003, i fornitori hanno server per le ordinazioni raggiungibili su internet, ma dietro un firewall, e così i clienti. Tutti questi sistemi, lontani geograficamente e disomogenei come tipologia di sistema operativo possono darci le informazioni e i servizi di cui abbiamo bisogno ma, utilizzando le tecniche esposte finora, dialogare con loro sarebbe un incubo. Cerchiamo di cambiare la nostra prospettiva: cominciamo a pensare queste parti in gioco non più in base a come sono realizzate o a quali astrazioni fanno riferimento, ma secondo quello che ci possono fornire, ovverosia cominciamo a pensarli come servizi di cui noi, o meglio il sistema informatico di cui andremo a definire l'architettura, saremo i fruitori. Ecco che nasce l'idea di un'applicazione veramente distribuita, le cui parti siano

1. fornitori di servizi, che possiamo sfruttare indipendentemente dal dove si trovano, dalla tecnologia su cui sono basati, dalla piattaforma software o hardware che li realizza

2. fruitori o consumatori di tali servizi.

QUESTIONE DI REGOLE

Chiaramente abbiamo bisogno di alcune regole che ci permettano di sfruttare questi servizi.

- Avremo bisogno di un linguaggio comune per poter far sì che i nostri fornitori di servizi possano capire le nostre richieste e che noi capiamo le loro risposte.
- Avremo bisogno che le regole stesse siano poche, chiare, fisse e persistenti. Che ci sia cioè un contratto stipulato tra il fruitore e il fornitore dei servizi, e che tale contratto sia facilmente comprensibile dalle macchine: non dovremo più condividere tipi e astrazioni; condivideremo regole validabili dalle macchine.
- Sarebbe interessante, poi, poter scoprire dinamicamente i servizi di cui abbiamo bisogno, in modo tale che la macchina che utilizza il servizio possa capire quali sono le regole da utlizzare.
- Per far questo chiaramente dovremo utilizzare dei messaggi su un canale di comunicazione, e sarebbe meglio che questo canale fosse sicuro, semplice e universalmente accessibile. Per esempio l'HTTP (HyperText Transfer Protocol) .

SOAP

SOAP: Simple Object Access Protocol. Un protocollo basato su XML (eXtensible Markup Language) per l'accesso remoto a oggetti software, introdotto fra gli altri da Microsoft. Sta diventando lo standard per l'accesso ai WebServices. Si veda www.w3c.org.

Quello che abbiamo descritto esiste, ed è la base della Service Oriented Architecture (SOA): un'architettura di applicazione in cui la parte di logica client e quella di funzionalità business sono disaccoppiate: quest'ultima è realizzata con dei moduli più semplici (i servizi), definiti formalmente da interfacce.

SOA: LE BASI

Fig. 1: **Struttura di un Web Service.**

Il linguaggio comune è l'XML, di cui possiamo utilizzare un dialetto, per esempio SOAP (Simple Object Access Protocol) o XML-RPC (XML Remote Procedure Calling) per chiedere e rispondere. L'utilizzo di SOAP su HTTP (ma non solo, si potrebbe usare SMTP, FTP...) è la base dei Web Services (WS), secondo quanto la maggior parte dell'industria del software ha ormai stabilito come standard di fatto. Lo standard di descrizione dei Web Service è il WSDL (Web Services Description Language). Una macchina può interrogare un fornitore di servizi, esaminare il file WSDL dato in risposta, e avere tutte le informazioni utili per interrogare il Web Service che le interessa, sapendone interpretare correttamente le risposte. Il sistema che potremmo utilizzare per scoprire le caratteristiche del Web Service di cui abbiamo bisogno, o addirittura per cercare in modo automatico un Web Service che risponda alle nostre esigenze è UDDI (Universal Description, Discovery and Integration). Abbiamo quindi delineato una nuova struttura architetturale orientata alla fruizione di servizi: SOA, appunto. In vediamo la struttura di un WebService, in cui un consumatore accede al fornitore di servizi, dove risiede sia il web service che la sua descrizione realizzata con WSDL. Il consumatore può anche cercare notizie sul Web Service presso una agenzia di discovery, dove il fornitore avrà pubblicato i suoi servizi. In Fig. 2 (struttura di una applicazione SOA) vediamo una possibile architettura basata sui servizi.

Fig. 2: **Struttura di una applicazione SOA.**

GLOSSARIO

WSDL

WSDL: Web Services Description Language
Un linguaggio che utilizza XML (eXtensible Markup Language) e fornisce la grammatica e il formato di specificazione delle interfacce esposte dai Web Service. È utilizzato per descrivere i Web Services specialmente in caso di pubblicazione.

EVOLUZIONE O RIVOLUZIONE?

Evoluzione e rivoluzione insieme: l'utilizzo dei Web Service fa sì che tutto quello che abbiamo sviluppato in questi anni non vada perso, ma diventi parte di processo dei Web Services: gli oggetti, i componenti diventano gli ingranaggi che fanno funzionare il motore dei Web Services, senza però che le astrazioni che realizzano debbano essere condivise. L'unica cosa condivisa è il contratto fra il service provider e il client. Quindi, più in dettaglio, nella prospettiva di una SOA:

1) Il servizio è la realizzazione di funzionalità software, e viene definito nei termini del contratto realizzato dalla sua interfaccia e non dall'astrazione dei suoi tipi. WSDL è il linguaggio utilizzato per la descrizione dei contratti. Il componente è scritto in modo da poter essere utilizzato solo in base al contratto (interfaccia) esposto.

2) I servizi sono coarse-grained, ovvero la granularità dei servizi più utili è grossa: un servizio generalmente fornirà informazioni ottenute dopo una elaborazione significativa. Nel nostro esempio, la lista dei prodotti in fine assortimento ordinata per prezzo e per magazzino. Servizi a granularità più fine, come quello che fornisce i prodotti per un singolo magazzino, vengono magari utilizzati dietro le quinte dal Web Service principale. Oggetti, componenti e servizi sono posti in scala crescente di utilità business e decrescente in granularità.

3) Il Web Service è auto-contenuto, anzi di più: il Web Service mantiene al suo interno il proprio stato. Generalmente le chiamate sono stateless ovvero non presuppongono che una chiamata già fatta modifichi lo stato del servizio.

4) Il Web Service è debolmente accoppiato con gli altri Web Service (loose coupling). Questo concetto è fondamentale, anche per considerazioni di scalabilità e di versatilità dell'applicazione.

5) Il Web Service può essere cliente e fornitore di altri Web Service.

6) I vari Web Service interagiscono scambiandosi messaggi su un canale virtuale detto message bus o service bus che può essere esterno all'azienda o interno. Tale canale non è necessariamente SOAP su HTTP. All'interno dell'azienda si potrebbe utilizzare un sistema di accodamento di messaggi e, se la comunicazione dei messaggi è punto punto, una buona scelta è XML-RPC.

7) L'estensibilità dei Web Service viene realizzata attraverso il routing dei messaggi.

8) I Web Service assumono che la connettività sia asincrona: alta latenza e alta rumorosità del canale non sono un problema. Il famoso sviluppatore/autore Don Box dice che questo concetto può anche estendersi alle persone coinvolte nella realizzazione dell'applicazione SOA: scambiandosi schemi XML e non tipi (come classi) si parla a "rumore zero". Il servizio quindi porta all'estremo il concetto di incapsulamento che tanto abbiamo amato scrivendo le nostre classi OO.

WEB SERVICES PER CHI?

Chi utilizza all'interno dell'impresa i vari Web Service? I processi. I processi fanno quella che viene chiamata orchestration dei Web Service: la logica di business utilizza i vari servizi per ottenere lo scopo voluto. Per rifarci all'esempio esposto prima sarà il nostro processo di business che chiederà al Web

Service del magazzino quali siano i prodotti da riordinare e, in base a queste informazioni, chiamerà il web service del fornitore tramite HTTP, effettuando l'ordine. Poi, all'interno della rete IT dell'azienda, il nostro processo chiamerà il Web Service della contabilità per informare dell'acquisto fatto, magari accodando la chiamata SOAP in MSMQ (Microsoft Message Queue Server). L'azione di sincronizzazione e di scambio di messaggi fra diversi gruppi di Web Service/ processi viene generalmente chiamata coreography. Chi ha esperienza di scrittura di applicazioni enterprise avrà intuito alcune delle problematiche nuove che una architettura di questo genere fa nascere. Per esempio nasce il bisogno di un nuovo modello di transazione. Nasce l'idea della long-term transaction, ovvero di una transazione di lungo termine, che coinvolge diverse entità, di cui non è possibile a priori determinare il tempo di risposta. Occorre quindi anche una nuova semantica delle transazioni, essendo il modello ACID (Atomicity, Consistency, Isolation, Durability) non del tutto adeguato.
Differenti iniziative sono nate attorno a questo problema, come Web Service Coordination (WS-C), Web Service Tansaction (WS-T) e il Business Transaction Protocol (BTP) di OASIS. Non solo l'orchestrazione ma anche la composizione dei Web Service è importante: un Web Service può fornire un unica interfaccia verso l'esterno per una serie di Web Service interni. (Per chi ama i pattern, si usa generalmente il pattern Facade). Quindi i Web Service da soli non risolvono tutti i problemi: è necessario uno strato di Integrazione dei servizi, che ne determini la descrizione, la scoperta (discovery), la composizione (composition) e la gestione nel processo e intraprocesso *(orchestration, coreography)* dei Web Service.

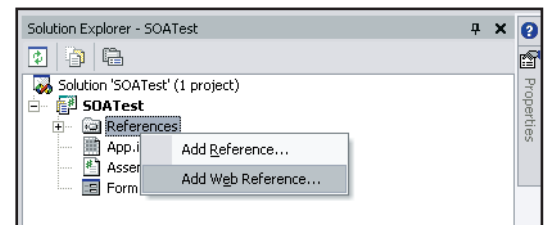
UN ESEMPIO DI APPLICAZIONE SOA

Esaminiamo un breve esempio. Scopo di questo esempio non è mostrare una applicazione di livello enterprise completa, non ce ne sarebbe lo spazio, ma illustrare come sia possibile utilizzare, integrare e coordinare Web Service da e su piattaforme e tecnologie differenti. Poniamo di voler creare una applicazione che ci aiuti ad migliorare le nostre conoscenze di programmazione. Vogliamo riunire in una sola applicazione quanti più servizi possibile, per esempio poter fare una ricerca per parole chiave in un sito di articoli tecnici e poter conoscere i dettagli dei libri che in quegli articoli vengono proposti. Questa è un'attività che in genere facciamo manualmente: per esempio possiamo utilizzare un browser per fare ricerche per parole chiave tra gli articoli di *http://dotnet.innovactive.it*, e magari poi andare su un sito di vendita di libri, per esempio *http://www.amazon.com* e ricercare i libri di cui si parla negli articoli, oppure effettuare la stessa ricerca fra i libri disponibili. Entrambi questi siti ci forniscono dei servizi, ma sono geograficamente remoti, non abbiamo accesso COM o CORBA ai loro componenti business, e comunque sono realizzati rispettivamente su Windows2003 e su UNIX, quindi su ambienti non compatibili. Potremmo automatizzare le nostre operazioni andando ad esaminare le pagine HTML con un parser ed effettuando le chiamate POST o GET, ma ogni cambiamento delle pagine renderebbe il nostro programma inutilizzabile. Fortunatamente entrambi questi due siti offrono un accesso limitato ai loro componenti business attraverso dei Web Service. Possiamo quindi utilizzare una qualunque tecnologia (per esempio Java o .NET) per consumare i servizi esposti e realizzare la nostra applicazione. Cominciamo da Amazon. Per prima cosa occorre scaricare dal sito l' Amazon.com Web Services software development kit (VEDI BOX "Il Web Service di Amazon"). Le informazioni sull'interfaccia, o meglio sul contratto cui i vari oggetti devono sottostare sono definite nel file WSDL pubblicato da Amazon, che troviamo online: *http://soap.amazon.com/schemas2/AmazonWebServices.wsdl*. Potremmo poi utilizzare java per interrogare il web service. Dal codice di esempio fornito con l'SDK (*AuthorSoap.java*) vediamo come utilizzare SOAP per fare una richiesta per autore, dopo che siano state generate le classi necessarie a partire dal file WSDL:

```
import java.net.URL;
import java.net.MalformedURLException;
import javax.xml.rpc.ServiceException;
import java.rmi.RemoteException;
public class AuthorSoap extends AbstractSoapQuery {
  public AuthorSoap() {
    super();
    this.parameters.put("Host",
                  "http://soap.amazon.com/onca/soap");
    this.parameters.put("Author","");
    this.parameters.put("Page","");
    this.parameters.put("Mode","");
    this.parameters.put("Tag","");
    this.parameters.put("Type","lite");
    this.parameters.put("Dev-Tag","");
    //this.parameters.put("Version",""); }
  public Object issueRequest() throws
            MalformedURLException, ServiceException,
                                RemoteException {
    com.amazon.soap.axis.AmazonSearchService
              service = new com.amazon.soap.axis.
```

Fig. 3: aggiunta di un riferimento Web.

UDDI

UDDI: Universal Description, Discovery and Integration Specifica per l'integrazione Business to Business (B2B) che permette ad entità come aziende di descrivere come intendono fare e-business, specialmente in termini dei Web Services esposti. Utilizza anche WSDL. Si veda www.uddi.org e uddi.microsoft.com

IL WEB SERVICE DI AMAZON

Amazon.com fornisce molteplici informazioni tramite un Web Service, che può essere utilizzato sia dai lettori che dai partner (venditori e compratori) di Amazon. Per sperimentare il Web Service di Amazon, occorre scaricare il Software Development Kit, e iscriversi gratuitamente al servizio, in modo da ottenere un "developer Token", ovvero una stringa di caratteri che autorizza l'uso del Web service. Tutto questo può essere fatto consultando la pagina: http://www.amazon.com/ e cercando il link "Web Services" Si raccomanda di leggere bene le condizioni d'uso del servizio.

```
                        AmazonSearchServiceLocator();
    com.amazon.soap.axis.AmazonSearchPort port =
                     service.getAmazonSearchPort(new
        URL((String)this.parameters.get("Host")));
    com.amazon.soap.axis.AuthorRequest request =
          new com.amazon.soap.axis.AuthorRequest();
    request.setAuthor((String)
                   this.parameters.get("Author"));
    request.setPage((String)
                     this.parameters.get("Page"));
    request.setMode((String)
                     this.parameters.get("Mode"));
    request.setTag((String)this.parameters.get("Tag"));
    request.setType((String)
                     this.parameters.get("Type"));
    request.setDevtag((String)
                  this.parameters.get("Dev-Tag"));
    //request.setVersion((String)
                  this.parameters.get("Version"));
    this.result = port.authorSearchRequest(request);
    return this.result;}
}
```

in questa classe vengono impostati i parametri della richiesta soap e il tipo di richiesta, viene istanziato un oggetto di tipo *com.amazon.soap.axis.AmazonSearchService*, viene istanziato un oggetto per la richiesta, e viene fatta la chiamata al Web Service: *authorSearchRequest*. La richiesta che abbiamo visto nell'esempio precedente è definita nel seguente frammento del file WSDL:

```
<xsd:complexType name="AuthorRequest">
  <xsd:all>
    <xsd:element name="author" type="xsd:string"/>
    <xsd:element name="page" type="xsd:string"/>
    <xsd:element name="mode" type="xsd:string"/>
    <xsd:element name="tag" type="xsd:string"/>
    <xsd:element name="type" type="xsd:string"/>
    <xsd:element name="devtag" type="xsd:string"/>
    <xsd:element name="sort" type="xsd:string"
                                  minOccurs="0"/>
    <xsd:element name="locale" type="xsd:string"
                                  minOccurs="0"/>
    <xsd:element name="keywords"
                   type="xsd:string" minOccurs="0"/>
    <xsd:element name="price" type="xsd:string"
                                  minOccurs="0"/>
  </xsd:all>
</xsd:complexType>
```

che costituisce il contratto tra il fornitore del servizio e il consumatore. Dato che possiamo compilare ed eseguire il codice Java sia sotto Windows che per esempio sotto Linux, e per entrambi la chiamata al Web Service non cambia, abbiamo un esempio della interoperabilità dei Web Service tra differenti sistemi operativi. Vediamo ora lo stesso esempio in .NET, aggiungiamo in questo caso qualche dettaglio in più, visto che nell'SDK di Amazon non è presente codice di esempio .NET. In questo esempio utilizzeremo con .NET anche il Web Service di dotnet.innovactive.it (vedi box "Il Web Service di InnovActive"), che ci permetterà di fare ricerche fra gli articoli pubblicati sul sito. In .NET se utilizziamo Visual Studio possiamo generare in modo del tutto automatico le classi locali che realizzano il "proxy" con il fornitore del Web Service. Chiaramente anche tali classi verranno generate a partire dal file WSDL che descrive il servizio: qualora invece non utilizzassimo Visual Studio, dovremmo passare il file WSDL all'applicazione wsdl.exe fornita con il .NET Framework SDK, e avremo generate le classi proxy. Apriamo Visual Studio, creiamo un'applicazione Winforms C# e facciamo click col pulsante destro del mouse sul progetto, scegliendo *Add Web Reference* come in Fig. 3 (aggiunta di un riferimento Web). Così facendo aggiungeremo un riferimento alle classi generate a partire dal contratto esposto dal Web Service: si aprirà una finestra in cui potremo immettere la URL del file WSDL voluto. Visual Studio effettuerà la connessione e ci presenterà i metodi forniti dal Web Service (metodi del web service di Amazon). Scegliendo Add Reference, verrà aggiunto il riferimento al Web Service nel nostro progetto. Esplorando infatti i namespaces del progetto tramite l'object browser (scorciatoia *ctrl-alt-J*) vediamo che sotto il namespace proprio della nostra applicazione viene mostrato il namespace *com.amazon.soap* con le classi che descrivono il Web Service. Possiamo utilizzare il Web Service di Amazon in modo molto simile a quanto già visto in java. In dettaglio vediamo del codice di esempio per utilizzare la ricerca per parole chiave in Amazon. Aggiungiamo alla form della nostra applicazione di test un Button che chiameremo *SearchAmazonBtn*, una TextBox in cui inseriremo le parole chiave, txtKeyWords, e una ListBox, *lbResults*, in cui verranno mostrati i risultati.
Innanzitutto importiamo il namespace:

```
using SOATest.com.amazon.soap;
```

nella nostra classe. Poi assegniamo un gestore all'evento click sul bottone di ricerca, e inseriamo il codice seguente, avendo l'accortezza di sostituire il developer Token ottenuto da Amazon nella riga di valorizzazione dell'attributo *keywordReq.devtag:*

```
private void SearchAmazonBtn_Click(object sender,
                          System.EventArgs e) {
  AmazonSearchService amSearchSvc= new
                        AmazonSearchService();
  KeywordRequest  keywordReq= new
                             KeywordRequest();
  keywordReq.keyword = txtKeyWords.Text;
  keywordReq.type = "heavy" ;
```

```
keywordReq.devtag = "TUO TOKEN";
keywordReq.mode = "books"   ;
keywordReq.page = "1";
try {
    ProductInfo pInfo=amSearchSvc.
                KeywordSearchRequest (keywordReq );
    lbResults.Items.Clear();
    foreach (Details Detail in pInfo.Details ) {
        lbResults.Items.Add(  Detail.ProductName +"\t"+
            Detail.Publisher +"\t"+Detail.Authors[0]); }
} catch (Exception Ex) {
    MessageBox.Show ("Errore nell'accesso al Web
                        Service: "+Ex.Message); }
}
```

il codice è estremamente semplice: prima viene inizializzato un oggetto di tipo *AmazonSearchService* poi uno di tipo *KeywordRequest*. Vengono poi valorizzati i campi di quest'ultimo oggetto, e viene eseguita la richiesta. Una volta ritornata la richiesta, se non c'è errore, vengono enumerati i risultati nel ciclo foreach, e vengono aggiunti alla lista dei risultati. Si noti come questo è solo un utilizzo di esempio del Web Service, ma basta per vedere come il modello a oggetti creato sia complesso e ricco di informazioni. Abbiamo fin qui utilizzato quindi un oggetto business che gira su un server Unix da un'applicazione Windows: utilizziamo ora la stessa tecnica con il Web Service realizzato in .NET da parte di InnovActive. Per Amazon abbiamo demandato a Visual Studio l'onere di importare il file WSDL nel nostro progetto e di generare le classi proxy: lo stesso può esser fatto con il servizio di ricerca di *http://dotnet.innovactive.it,* il cui file WSDL è scaricabile a *http://ws.innovactive.it/innovActiveWS.asmx?WSDL*.

In Fig. 4 (*Namespaces* del progetto di test) si vede come i due Web Service siano stati integrati nel progetto. Si noti che il nome del namespace viene proposto dalla finestra di importazione del file WSDL, ma può essere cambiato a piacimento: per questo il nome che verrà visualizzato nei vostri progetti di prova non rispecchierà necessariamente quello esposto in questo articolo. Analogamente a quanto visto sopra, per non dover specificare il nome qualificato degli oggetti ogni volta che verranno utilizzati, dovremo specificare l'uso del namespace

```
using SOATest.ws.innovactive.it ;
```

per la nostra classe. Allo stesso modo di prima creeremo un bottone per la ricerca, ma utilizzeremo la stessa TextBox e ListBox:

```
private void btnInnovactiveSearch_Click(object sender,
                            System.EventArgs e) {
  InnovActiveWS myWebService= new InnovActiveWS();
  Article[] ArrayOfArticles= myWebService.SearchArticles
    (txtKeyWords.Text ,10,SearchTypeEnum.EXACT_PHRASE );
  try {
    foreach (Article Art in ArrayOfArticles )
    {
      string s1=String.Format("[{0}] {1}\tby {2}, URL: {3})",
      Art.Category,Art.Title,Art.Author,Art.URL);
      lbResults.Items.Add(  s1); }
  }
  catch (Exception Ex)
  {
    MessageBox.Show ("Errore nell'accesso al
                       WebService: "+Ex.Message);
  }
}
```

in questo caso vediamo come viene creato un oggetto che rappresenta il Web Service, e come viene chiamato il metodo SearchArticles del Web Service. È interessante notare che il terzo parametro della chiamata è un membro di una enumerazione (*SearchTypeEnum.EXACT_PHRASE*): infatti nel file WSDL e quindi nelle classi generate è possibile anche specificare tipologie di dati complesse come le enumerazioni:

```
<s:simpleType name="SearchTypeEnum">
  <s:restriction base="s:string">
    <s:enumeration value="EXACT_PHRASE" />
    <s:enumeration value="ALL_WORDS" />
    <s:enumeration value="ANY_WORD" />
  </s:restriction>
</s:simpleType>
```

Abbiamo quindi integrato in una applicazione due Web Service diversi, utilizzando il contratto da essi esposto come unica sorgente di informazioni sul loro utilizzo. Possiamo anche aggiungere un terzo Button che faccia la ricerca in entrambe le sorgenti ed elabori in qualche modo i due insiemi di risultati trovati (per esempio potrebbe far sì che cliccando su di un risultato si apra un browser che mostri l'articolo o il libro selezionato, o potrebbe fare ricerche incrociate): abbiamo così realizzato un esempio di logica business che fa una orchestrazione dei due Web Service.

CONCLUSIONI

La connessione di macchine differenti attraverso una rete è la caratteristica principale dell'informatica moderna: l'architettura orientata ai servizi porta questo concetto all'estremo, creando le premesse per una nuova tipologia di applicazioni business dove finalmente il servizio fornito e non la tecnologia sottostante è la chiave di valutazione delle infrastrutture software.

Marco Poponi

Objects
- mscorlib
- system.data
- system
- system.drawing
- system.web.services
- system.windows.forms
- system.xml
- soatest
  - {} SOATest
    - Form1
  - {} SOATest.com
  - {} SOATest.com.amazon
  - {} SOATest.com.amazon.soap
  - {} SOATest.ws
  - {} SOATest.ws.innovactive
  - {} SOATest.ws.innovactive.it
    - Article
    - Author
    - InnovActiveWS
    - SearchTypeEnum

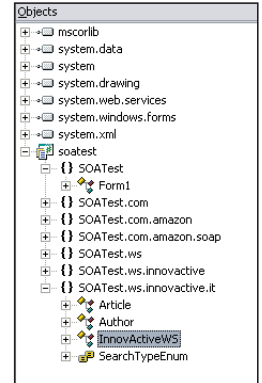



Fig. 4: Namespaces del progetto di test.

dotnet.InnovActive.it fornisce un Web Service ai lettori di Io Programmo in modo che possano sperimentare con i Web Service. Il web service è realizzato su piattaforma .NET, e fa riferimento agli articoli sullo sviluppo .NET pubblicati nel sito

http://dotnet.innovactive.it

con ricerche per parole chiave, autore ecc. Informazioni, condizioni d'uso e stato di rilascio possono essere trovati su

http://ws.innovactive.it

Il file WSDL del servizio si trova all'URL:

http://ws.innovactive.it/innovActiveWS.asmx?WSDL

Estendere le funzionalità di Visual Studio.NET

In questo articolo vedremo come creare, da zero, un Add-In per Visual Studio.NET, ovvero un programma di utilità che si integri perfettamente all'interno del moderno IDE di Visual Studio.NET.

Creeremo un'applicazione che ci permetta di effettuare inserimento di codice all'interno dei documenti aperti e che esponga un'interfaccia utente da noi progettata all'interno di Visual Studio. Seguitemi, dunque, perché le cose da dire sono molte…

VISUAL STUDIO.NET: FUCINA DI TECNOLOGIA

Se avete già cominciato a lavorare con Visual Studio .NET, vi sarete resi conto che tutto l'IDE è permeato di una tecnologia molto avanzata. Anche se tutti vi avranno detto che l'IDE è semplicemente un editor di codice, il che è in parte vero, c'è molto di più nascosto sotto la scorza che ne rappresenta la facciata. La possibilità di integrare dei plugin ad-hoc per le nostre necessità, è ancora più facile che nelle versioni precedenti di Visual Studio. È noto che risulta un'impresa impossibile produrre un IDE che soddisfi tutte le necessità degli sviluppatori, e Visual Studio non fa certo eccezione, ma ci viene in aiuto con un modello di estensibilità veramente potente e flessibile. I cosiddetti "Add-In" inoltre, possono essere sviluppati con i linguaggi del Framework, C# o VB.NET, non necessitando di particolari concetti, se non quelli di base riguardanti appunto il modello di estensibilità di Visual Studio .NET.

Fig. 1: *Il modello di automazione di Visual Studio .NET.*

MACRO O ADD-IN?

VS.NET mette a disposizione dello sviluppatore due strumenti per l'estensibilità: le macro e gli Add-In. Le prime sono ciò che dice la parola stessa: macro-comandi, ovvero batch di comandi impartiti all'IDE per effettuare azioni ripetitive o ricorrenti. Pensate ad esempio all'azione di attivare il debug sul processo *aspnet_wp.exe*: di norma sono richiesti un click sul menu, la selezione di una voce, la scelta del processo dalla lista dei processi attivi e la successiva pressione di tre pulsanti. L'effetto è sempre lo stesso, e quindi è possibile pensare ad una macro che raccolga i comandi in un'unica pressione di un tasto. Le macro sono dei file con estensione *.vsmacro*, sono in chiaro (non compilate) e necessitano solo di un doppio click per essere attivate. Sono pertanto la scelta migliore per soluzioni di estensibilità relativamente semplici. Gli Add-In sono invece dei veri e propri programmi compilati, con una loro interfaccia utente, che devono essere distribuiti attraverso un package di installazione *.msi*. La loro robustezza e flessibilità li rende adatti per effettuare compiti più articolati di quelli realizzati tramite macro. In que-

sto articolo non affronteremo l'utilizzo di macro, bensì andremo a sviluppare un Add-In per l'aggiunta dinamica di commenti e codice vario all'interno dei nostri progetti.

LA CREAZIONE DELL'ADD-IN

La creazione dell'Add-In è stata resa estremamente semplice in VS.NET, essendo presente nell'elenco delle tipologie di progetti il template *Visual Studio.NET Add-In*. Il template di progetto si preoccupa di creare tutto il plumbing necessario ad effettuare l'interoperabilità con COM ed importa automaticamente le referenze necessarie per interagire con l'IDE di Visual Studio.NET. La finestra di creazione progetto può essere osservata in Fig. 2.

***Fig. 2:** Creiamo il progetto del nostro Add-In in modo del tutto automatizzato.*

La nostra applicazione si chiamerà *Comment-O-Matic* e permetterà l'inserimento di codice di commento, Regioni di codice ed altre funzionalità all'interno dei nostri progetti. Impostiamo quindi il nome nella prima pagina di creazione progetto e avviamo il Wizard. Andremo ora ad elencare tutte le azioni che dovremo fare per impostare il nostro Add-In tramite il Wizard:

Come linguaggio, scegliamo Visual Basic. Può essere ovviamente utilizzato anche C#, ma visto che personalmente sono un estimatore di VB, impiegherò quest' ultimo per sviluppare l'Add-In;
Togliamo il check dall'applicazione VSMacro, in modo da non permettere all'Add-In di essere ospitato dall'applicazione per la creazione di macro, bensì solo dall'IDE di VS.NET.
Aggiungiamo un nome ed una descrizione per il nostro Add-In. Il nome e la descrizione sono importanti e verranno visualizzati nella finestra di dialogo degli *Strumenti Aggiuntivi* di VS.NET, reperibile attraverso il menu *Tools>>Add-In Manager* (oppure *Strumenti>>Componenti Aggiuntivi* se si sta usando VS.NET in Italiano).
Nella pagina 4, impostiamo il flag che indica che il

***Fig. 3:** Impostiamo i flags del wizard per ottenere i risultati voluti.*

nostro Add-In creerà un elemento all'interno del menù strumenti. Questo sarà utile per attivare il nostro Add-In come vedremo in seguito. Impostiamo anche il flag che rende disponibile l'Add-In a tutti gli utenti del computer.
Nell'ultima pagina possiamo scegliere se offrire una *"About Window"* nella quale rendere pubblici gli estremi del creatore o della ditta creatrice dell'Add-In.

Una volta completato il wizard, Visual Studio.NET creerà per noi una serie di file, progetti e referenze utili per la creazione del nostro Add-In.
Andiamo allora ad esaminarli in dettaglio.

PROGETTI, FILE E REFERENZE

Il nostro prode wizard ha fatto per noi un lavoro veramente notevole! Come potete vedere in Fig. 4, è stata creata per noi una soluzione contenente due progetti: uno è il progetto vero e proprio del nostro Add-In, mentre l'altro è il progetto di distribuzione, ovvero il progetto che si occupa di prendere l'Add-In compilato e di creare il package di Windows Installer (msi) che ci permetterà di distribuire l'Add-In ai nostri colleghi. Vi invito a riflettere su quanto codice VS.NET ha creato per noi e su quanto tempo andremo a risparmiare proprio grazie a questo fatto. Tra le referenze importate nel nostro progetto Add-In, possiamo notare 3 dipendenze inusuali, ovvero:

- **EnvDTE:** sta per *Design Time Environment*, ovvero è la referenza all'ambiente di sviluppo stesso.
- **Extensibility:** contiene l'interfaccia *IDTEExtensibility2* che deve essere implementata da ogni Add-In che creeremo;
- **Office:** contiene le interfacce e gli *enum types* utili per accedere ai menu ed alle toolbar dell'applicazione host, ovvero VS.NET.

CODE GENERATION

Vi siete mai chiesti come fa Visual Studio a creare il codice mentre noi ce ne stiamo beati a disegnare i controlli dalla parte "design" delle Windows Forms? Impiega uno dei meccanismi più complicati ed articolati del .NET Framework, ovvero il *System.CodeDom*. Questo namespace contiene le classi atte a generare il codice nel modo canonico ed indipendente dal linguaggio. Grazie al loro impiego è possibile emettere codice e compilarlo dinamicamente, ovvero creare programmi che creano altri programmi! Forse Matrix non è molto lontana...

***Fig. 4:** L'output generato dal wizard per gli Add-In di Visual Studio.NET.*

L'unica classe che il wizard ha creato per noi è la *connect.vb*. Questa classe rappresenta il vero e proprio "innesco" del plugin all'interno di VS.NET. Vediamo in dettaglio ciò che deve essere implementato per far funzionare correttamente il tutto:

- **Connect.New:** è il costruttore della classe *connect*. Qui possono essere fatte le inizializzazioni del nostro Add-In.
- **Connect.OnConnection:** Il metodo *OnConnection* viene richiamato all'effettivo caricamento dell'Add-In all'interno dell'IDE. In questo metodo, peraltro già popolato dal wizard, sarà nostra cura informare l'IDE dei comandi disponibili, i binding di tastiera e le eventuali finestre da creare all'interno dell'ambiente di sviluppo.
- **Connect.QueryStatus:** il metodo viene richiamato ogniqualvolta l'ambiente di sviluppo deve determinare se un particolare comando è o meno disponibile nello stato corrente.
- **Connect.Exec:** questo metodo viene invocato dall'IDE per eseguire materialmente i comandi.

Gli altri metodi creati dal Wizard, necessari per la corretta implementazione dell'interfaccia *IDTEExtensibility2* sono meno interessanti e non direttamente interessati alla creazione del nostro semplice Add-In. Vediamo ora di implementare la nostra prima semplice funzionalità: la creazione di una riga di commento.

IL NOSTRO PRIMO ADD-IN

Cominciamo con un esercizio molto semplice, la creazione di una riga di commento. Questo esempio ci permetterà di spiegare il comportamento dell'Add-In e come quest'ultimo reagisce alle azioni dell'utente. Nel corpo della Classe *Connect* creiamo la funzione *WriteCommentLine* che accetta in input una stringa e da come output un booleano.

```
Private Function WriteCommentLine(ByVal
                commentLine As String) As Boolean
If Not (applicationObject.ActiveDocument Is Nothing) Then
CType(applicationObject.ActiveDocument.Selection,
                              TextSelection).Text
= "'" & commentLine & vbCrLf
Return True
End If
End Function
```

Come far scattare questa routine? Utilizzando i comandi del nostro Add-In. I comandi rappresentano le "leve" sulle quali si può agire dall'esterno, ovvero dall'IDE, dai menu, dalle toolbar e persino dalle finestre di comando *(!!)* di VS.NET. I comandi sono implementati nel metodo *OnConnection*, ed il wizard ne ha creato per noi uno di default, con il nome di *CommentOMatic*. È di fondamentale importanza distinguere il ruolo della routine *OnConnection* da quello della routine *Exec*. *OnConnection* crea i comandi, marcandoli ed inserendone le descrizioni, mentre Exec si occupa di eseguirli, ovvero una volta che questi vengono invocati dall'IDE, questa routine esamina il *"CommandName"* ed intraprende le azioni appropriate.

```
Dim CommandObj as Command
CommandObj = applicationObject.Commands.
    AddNamedCommand(objAddIn, "CommentOMatic",
        "CommentOMatic", "Executes the command for
           CommentOMatic", True, 59, Nothing, 1 + 2)
'1+2 == vsCommandStatusSupported+
                         vsCommandStatusEnabled
```

Il comando creato dal Wizard verrà riconosciuto come *CommentOMatic.Connect.CommentOMatic*, e potrà essere invocato dall'IDE attraverso la finestra di comando, addirittura facendo uso dell'intellisense *(!)*. Come fare per provarlo? Dobbiamo modificare all'interno della routine *Exec* la riga *handled=true* in *handled=WriteCommentLine("This is a comment")*. Ora, premendo *F5* si aprirà una nuova istanza di Visual Studio .NET, con il Debug e tutto il resto, la quale ci permetterà di testare il nostro Add-In. Troveremo l'Add-In all'interno del menù strumenti (Tools). Se non compare la voce *CommentOMatic*, provate a lanciare il file *.reg* presente all'interno della directory dove risiede il codice dell'Add-In. Quando l'Add-In viene rilevato da VS.NET, comparirà una simpatica faccina tipo smile con la scritta *CommentOMatic* a fianco. A questo punto, aprite un qualsiasi documento di testo, ad esempio una classe, all'interno dell'IDE ed aprite anche la finestra di Comando *(Command Window)*. Sia cliccando la voce di menu che scrivendo per esteso il comando nella command window, si ottiene lo stesso risultato, ovvero quello di inserire una riga di commento all'interno del file correntemente aperto. Ovviamente, il nome del comando non è molto sensato e si potrebbe cambiarlo da *CommentOMatic* in *WriteCommentLine;* come fare? Semplice: guardate il

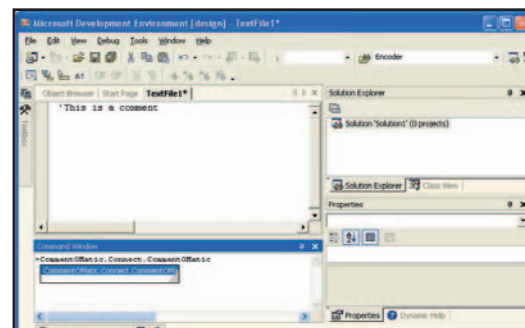



Fig. 5: ***Il nostro Add-In al lavoro.***

NOTA

VISUAL STUDIO.NET

Ancora una volta mi trovo a dover spezzare non una, bensì molte lance a favore di questo splendido ambiente di lavoro. In questo articolo abbiamo visto come anche Visual Studio, come il fratello .NET Framework, sia stato sviluppato con in mente l'estendibilità delle funzioni. Se ciascuno di voi si fermasse a pensare a quante funzionalità gli potrebbero essere utili mentre lavora con Visual Studio.NET, potrebbe realizzare molti Add-In utili non solo a se stesso, ma a tutta la comunità degli sviluppatori.

codice sotto riportato…

```
Dim CommandObj as Command
CommandObj = applicationObject.Commands.
  AddNamedCommand(objAddIn, "WriteCommentLine",
      "Write a comment line", "Writes a comment line ",
                              True, 59, Nothing, 1 + 2)
'1+2 == vsCommandStatusSupported+
                              vsCommandStatusEnabled
'Nei metodi QueryStatus ed Exec, cambiare la
                                       seguente riga
If cmdName = "CommentOMatic.Connect.
                                  CommentOMatic" Then
'sostituendola con
If cmdName = "CommentOMatic.Connect.
                               WriteCommentLine" Then
```

Questo tanto per capire quanto sia semplice cambiare o aggiungere comandi al nostro Add-In. Ricordatevi però che i cambiamenti apportati ai comandi non avranno effetto fino a che non ricompilerete entrambi i progetti, quello dell'Add-In e quello di installazione e non procederete all'installazione dell'Add-In tramite il proprio programma di installazione. Questo perché il wizard carica i dati per l'esecuzione dal registro di configurazione la prima volta, ovvero quando fate partire il processo di creazione dell'Add-In, ma non le volte successive. Se avviate il file *ReCreateCommands.reg* trovate i nuovi comandi, ma anche quelli vecchi, non più disponibili, per cui vi consiglio di pianificare attentamente l'interfaccia di comando del vostro Add-In, in modo da minimizzare il numero di volte il cui dovrete disinstallare l'Add-In e reinstallarlo. L'unico modo di capire come funziona la logica di installazione degli Add-In all'interno di Visual Studio consiste nello sperimentare, sperimentare, sperimentare! Le prime volte sarà frustrante, molte volte i file saranno bloccati e non potrete sovrascriverli, ma poi con la pratica tutto verrà naturale.

L'ADD-IN È DAVVERO INTEGRATO!

Non è magia, è solo merito dell'automazione offerta da Visual Studio.NET. Se provate ad aprire l'object Browser di VS.NET, e cercate l'interfaccia _DTE, scoprirete quante proprietà e metodi utilissimi esponga quest'oggetto, che nel nostro Add-In e referenziato dalla variabile applicationObject. Questa quindi rappresenta di fatto l'applicazione ospitante il nostro Add-In, ovvero VS.NET. La variabile *applicationObject* viene inizializzata nel metodo *OnConnection*, nello stesso momento in cui viene avviato Visual Studio e vengono dinamicamente collegati gli Add-In resi disponibili nella finestra di dialogo Add-In Manager. È bene pensare all'oggetto ospitante come ad un "fornitore di servizi" al quale il nostro Add-In si appoggia per eseguire i compiti per cui è stato progettato. Un altro esempio di interazione tra l'Add-In e l'ambiente di sviluppo si può vedere nella riga dove viene invocato il documento selezionato e la selezione corrente; questa è di fatto un'interrogazione dell'ambiente di sviluppo effettuata per determinare un riferimento ad oggetti che non appartengono all'Add-In, bensì all'IDE di Visual Studio.NET. Risulta essere molto utile la possibilità di collegare tasti di scelta rapida per invocare i comandi dell'Add-In, proprio come se fosse un comando nativo di Visual Studio, ed in più è possibile collegare il comando creato ad una gerarchia di menu preesistente. A questo scopo, analizziamo la Helper Function *AddCommandEx* che semplifica la creazione di comandi di un Add-In. Nella *AddCommandEx* si possono notare diverse tecniche per l'integrazione dei comandi all'interno di Visual Studio. I parametri possono spiegare molte cose

- **rootName:** è la base del nome del comando. Solitamente è formata da <*namespace*>.*Connect.*
- **commandName:** è il nome del comando. Assieme al *rootName* contribuisce a rendere univoco il nome del comando all'interno dell'editor di VS.NET.
- **text:** è il nome del comando scritto nel menù, ovvero il *Friendly Name.*
- **toolTip:** penso che la parola lo descriva da sola. Rappresenta il suggerimento che appare quando si fa sostare il puntatore del mouse sul comando.
- **key:** rappresenta un key binding di default assegnato al comando, nel nostro caso impostato a *Global::Alt+c,Alt+i*. Per vedere i binding, modificarli o aggiungerli, visitate il menu *Tools>>Options>>Keyboard.*
- **parentMenu:** rappresenta il menu che dovrà contenere il comando. Possiamo impostarlo liberamente, sapendo che il comando potrà comunque essere spostato successivamente grazie alla funzione personalizza di VS.NET. Sarà anche possibile spostare il nostro comando su una toolbar.

```
Private Sub AddCommandEx(ByVal rootName As String,
      ByVal commandName As String, ByVal text As String,
         ByVal toolTip As String, ByVal key As String, ByVal
                                  parentMenu As String)
Dim _cmd As Command
Dim _fullCommandName As String = rootName & "."
                                      & commandName
Dim GuidArray() As Object = {}
Try
'Se il comando esiste già, non lo devo aggiungere
                                         nuovamente...
_cmd = applicationObject.Commands.Item(
```

AUTOMATIZZARE LE AZIONI RIPETITIVE

Molti dicono che i programmatori sono pigri… beh, non abbastanza! Ogni volta che vi trovate a ripetere troppe volte una stessa azione, dove nell'esecuzione della stessa potreste commettere degli errori che potrebbero pregiudicare il buon funzionamento del software che state realizzando, è il momento di pensare se sia o meno il caso di realizzare un Add-In che ne automatizzi l'esecuzione. Non scrivete mai due volte la stessa cosa!!

```
                                        _fullCommandName, -1)
Catch ex As System.ArgumentException
'do nothing
End Try
If _cmd Is Nothing Then
'Creo il comando
_cmd = applicationObject.Commands.AddNamedCommand(
       addInInstance, commandName, text, toolTip, False, 1,
       GuidArray, 0) '1+2 == vsCommandStatusSupported+
                                        vsCommandStatusEnabled
End If
If key <> String.Empty Then
If _cmd.Bindings.Length >= 0 Then
Dim _bindings(0) As Object
_bindings(0) = CType(key, Object)
_cmd.Bindings = _bindings
End If
End If
If parentMenu <> String.Empty Then
'Se specifico il menu padre, aggancio il comando al
                                        menu (commandBar)
Dim _cmdBar As CommandBar = CType(applicationObject.
              CommandBars(parentMenu), CommandBar)
  _cmd.AddControl(_cmdBar, _cmdBar.Controls.Count)
End If
End Sub
```

A questo punto, molti di voi si chiederanno: tutto questo sforzo per una linea di commento? Eh no, adesso arriva il bello! Tutto quello che avete visto fin qui è privo di interfaccia utente, ovvero si tratta di Add-In tranquillamente realizzabili anche con delle macro. Ora cominceremo a fare sul serio, creando una nostra ToolWindow ed integrando un controllo utente all'interno di Visual Studio.

VISUAL INTEGRATION

Avete presente la finestra dove si trova la soluzione correntemente aperta in Visual Studio, oppure la finestra delle proprietà di un controllo WinForm, o ancora la finestra contenente il class View? Bene, esse sono dette "ToolWindows" e possono essere "sganciate" dall'ambiente di sviluppo, rese flottanti, riagganciate e rese auto-minimizzanti. Bello eh? E se potessimo farlo anche noi? Ovvero se potessimo scrivere un controllo ed incorporarlo all'interno di una nostra ToolWindow, beneficiando di tutti i vantaggi offertici dall'IDE di Visual Studio? Come dite? Non è possibile? E invece si! Ed è anche semplice!
Arricchiamo il nostro progetto di un semplice controllo utente che chiamiamo *ctlCommentOMatic*. Per il momento non aggiungiamo controlli al nostro User Control; piuttosto facciamo dei test per capire come integrarlo all'interno dell'interfaccia di VS .NET. Vogliamo caricare questo controllo all'interno di una ToolWindow, in modo da dare ai nostri utenti un'interfaccia per inserire i commenti più flessibile e ricca di quella offerta da un semplice comando su un menu. Esaminiamo dunque le due funzioni seguento, presenti nel codice incluso nel CD:

- *Private Function CreateAndDockToolWindow()*
- *Private Sub AddToMainWindow(ByVal win As Window).*

La helper function *CreateAndDockToolWindow*, crea una ToolWindow ospitante un controllo di tipo *CommentOMatic.ctlCommentOMatic*. Il metodo *CreateToolWindow* infatti riceve in input come parametri:

- L'istanza dell'Add-In;
- Il "Class Name" del controllo da creare;
- Il Titolo della ToolWindow
- Un GUID (univoco) che identifichi la finestra creata all'interno dell'interfaccia di Visual Studio.NET.

Di ritorno abbiamo un riferimento alla finestra appena creata ed un riferimento all'istanza del controllo appena integrato. Il riferimento può essere utilizzato per impostare varie proprietà del controllo utente e per effettuare delle operazioni di inizializzazione. A questo punto la finestra ed il controllo sono stati creati; quello che manca è agganciarli all'interfaccia principale e renderli visibili. Di queste operazioni si occupa la funzione AddToMainWindow, la quale, ottenuto un riferimento alle Linked Windows (*lw* nel codice), aggiunge all'insieme anche la nuova finestra. L'unica differenza tra l'Add-In senza interfaccia e quest'ultimo è la chiamata nel metodo Exec. Infatti mentre prima si invocava la semplice funzione *WriteCommentLine*, ora si invoca la *CreateAndDockToolWindow*. Riassumendo quindi, abbiamo visto come con due semplicissime funzioni, sia possibile integrare un nostro controllo all'interno di VS.NET. Non ci resta che premere *F5*, e quando andremo a cliccare sul comando "Write Comment Line", invece di ritrovarci con una linea di commento scritta all'interno del documento attivo, ci troveremo con una nuova finestra (grigia, in quanto il controllo non contiene controlli figli) di nome *CommentOMatic* agganciata all'IDE di Visual Studio.

UN CONTROLLO PRODUTTIVO

Tutto quello che abbiamo visto è molto educativo, ma dovremmo far fare qualcosa di interessante al nostro controllo per renderlo produttivo; in questa ultima parte della nostra analisi faremo generare al nostro controllo:

RAZIONALIZZARE IL MODO DI SCRIVERE

Un altro vantaggio derivato dall'impiego di Add-In nel processo produttivo è la razionalizzazione e la canonizzazione del codice. Se voi ed i vostri colleghi impiegaste un Add-In per scrivere le classi di accesso ai dati, ad esempio, non vi trovereste con due classi dove in una la gestione delle eccezioni è presente, mentre nell'altra risulta non essere stata implementata. Anche lo stile di scrittura del codice dei vari sviluppatori subisce una razionalizzazione, indotta appunto dall'uso degli Add-In.

- **Commenti**
- **Regioni**

D'ora in poi, gran parte dei concetti saranno relativi alla programmazione con Windows Forms, con qualche excursus nel modello ad oggetti di Visual Studio.NET. Cominciamo con l'interfaccia utente; come potete vedere in Fig. 6, il controllo è piuttosto spartano, ma funzionale. Ovviamente se avessimo voluto costruire qualcosa di più articolato, sarebbe stato d'obbligo pensare a qualcosa di più ergonomico, ma per i nostri scopi andrà più che bene così.
Possiamo vedere due *TabPages*, una per ogni funzione implementata dal nostro Add-In. Voglio farvi notare che è sempre bene organizzare le interfacce degli Add-In con controlli *Panel, Toolbar* o comunque container, in modo da renderli dockable e permettere così una gestione dell'interfaccia utente coerente con quella di Visual Studio. Nel nostro caso ho impostato *dock=fill* per il controllo *TabStrip* ed ho ancorato i vari controlli *TextBox* e *Button*. Una delle prime cose che vanno fatte è creare una proprietà del nostro controllo per memorizzare un riferimento all'*applicationObject*, in quanto il nostro controllo farà uso delle proprietà dell'ambiente integrato. Pertanto, dopo aver creato un riferimento al Namespace *EnvDTE* all'interno del nostro controllo,procediamo con la creazione della property *DTEObject*.

```
Imports EnvDTE
...
Private _oDTE as EnvDTE._DTE
...
Public Property DTEObject() As DTE
Get
Return _oDTE
End Get
Set(ByVal Value As DTE)
_oDTE = Value
End Set
End Property
```

Questa property andrà inizializzata appena dopo la chiamata alla funzione *CreateAndDockToolWindow*, all'interno del metodo *Exec*. A questo punto, il nostro controllo è pronto per interagire con l'IDE di Visual Studio.NET. Cominciamo con la funzione più semplice: Inserisci Commento, ovvero *Commentize* per gli amici!

INSERIMENTO DEI COMMENTI COMMENTIZE

Visto che siamo con "le mani in pasta", miglioriamo l'inserimento di commenti, abilitando il riconoscimento del linguaggio del file che stiamo commentando. Inseriamo una funzioncina privata che rileva il tipo di file attualmente selezionato e che permette un inserimento dei commenti corretti. Sappiamo tutti, infatti, che mentre Visual Basic impiega l'apice per commentare, C# impiega la doppia barra e l'asterisco barra (// e /*…*/). I file HTML/XML, invece impiegano la notazione (<!—…—>). Altro non c'è da dire, se non osservare direttamente il codice richiamato alla pressione del pulsante "Insert Comments", presente nel CD: *Private Sub Commentize(ByVal comments As String).*

LA FUNZIONE REGIONIZE

La funzione "Regionize" deve permettere la creazione di una Region ex novo ed inoltre deve contemplare la possibilità di voler "regionizzare" una selezione di testo. Tutti sappiamo cosa sono le region di Visual Studio.NET: sono delle innocue righe del tipo (#Region "Nome regione"…#End Region) che servono a mettere un po' di ordine nel codice sorgente. Dovremo lavorare un po' con il modello ad oggetti di VS.NET, per creare le regioni proprio come desideriamo, ma il risultato consisterà in una funzione Regionize che ci darà la possibilità di selezionare del testo, per esempio tutte le proprietà pubbliche di una classe, e racchiuderle all'interno di una Regione di nome "Proprietà pubbliche". Come fare? Vediamo il codice di *Private Sub Regionize(ByVal regionName As String)*, che trovate all'interno del progetto allegato al CD. Si è dovuto impiegare il modello ad oggetti di VS.NET perché se si scrive la parola #Region "nome regione" e si preme il tasto invio, l'editor crea automaticamente la corrispondente end region, cosa che noi non vogliamo. Per ovviare a questo problema abbiamo memorizzato la prima e l'ultima linea della selezione corrente e poi, manualmente siamo andati prima a scrivere la "end region" e poi la "Region", "bidonando" così l'editor che in questo caso non prende iniziative di completamento del codice.

CONCLUSIONI

I nostri Add-In sono strumenti che ci permettono di sviluppare più in fretta, meglio, e in modo più accurato i nostri progetti, automatizzando operazioni noiose e ripetitive. Essi ci permetto di conservare il nostro tempo e di impiegarlo meglio: magari tirando fuori delle fantastiche idee per le nostre future implementazioni. Sempre da realizzare con .NET, ovviamente! Alla prossima, e buon lavoro!

G. Davide Senatore

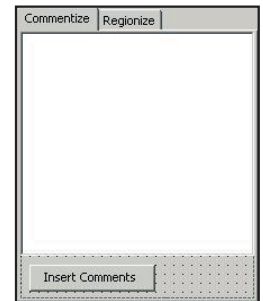



Fig. 6: L'interfaccia utente dell'Add-In.

PROGETTARE GLI ADD-IN

Quando progettate un Add-In, abbiate sempre chiaro in mente ciò che volete ottenere e da quali input dovete partire. Immaginate l'Add-In come una scatola nera che riceve in input determinati dati, accetta dei parametri ed elabora i dati in ingresso tenendo conto delle condizioni al contorno. Il risultato dovrà essere infallibile e deterministico.

Biblioteca

ON LINE

JAVA.SUN.COM

La Sun ha riprogettato il suo portale dedicato a Java. Vale proprio la pena di dargli un'occhiata.

http://java.sun.com/

DELPHI SOURCE

News, componenti, libri, Link, tips, dedicati al linguaggio principe fra gli ambienti di sviluppo RAD.

http://www.delphisource.com/

JSP INSIDER

Una manna da cielo per tutti gli sviluppatori di pagine JSP (Java Server Pages). Una montagna di informazioni e risorse che ogni buon programmatore non può far a meno di visionare.

http://www.jspinsider.com/

MANUALE PRATICO DI JAVA – SECONDA EDIZIONE

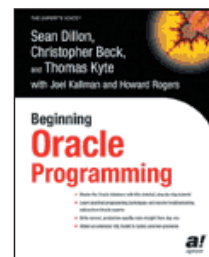
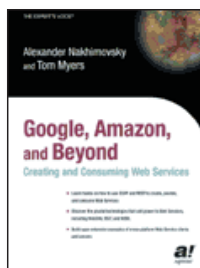
Una guida al linguaggio Java che racchiude le nozioni base, ma non solo. Gli autori si sono impegnati a fondo cercando di utilizzare un linguaggio decisamente comprensibile e non ostico. Ciò che si richiede al lettore poco esperto è una certa dose di impegno per apprendere i passaggi più complessi. L'organizzazione modulare dei vari capitoli fornisce il percorso migliore per passare dagli argomenti più semplici a quelli più avanzati. Tra gli argomenti trattati:

- Processi e multitasking
- Input/Output
- Java e i database
- Java Beans
- Java e XML
- Servlet API
- Java Server Pages

Difficoltà: Bassa • Autori: A.Gini, P.Aiello, L.Bettini, G.Puliti
Editore: HOPS – Tecniche nuove http://www.hopslibri.com • ISBN: 88-481-1553-5
Anno di pubblicazione: 2003 • Lingua: Italiano • Pagine: 360 • Prezzo: € 34,90

BEGINNING ORACLE PROGRAMMING

Questo testo rappresenta una vera e propria bibbia per chi si accinge ad apprendere la programmazione dell'RDBMS più famoso al mondo: Oracle. Il libro mostra le tecniche, gli strumenti, e le diverse tipologie di programmazione, utilizzate sia da sviluppatori che da amministratori di sistema. Il processo d'apprendimento è graduale, in modo da dare al lettore, che non ha mai utilizzato Oracle, uno valido strumento d'addottrinamento.
Gli esempi a corredo del testo forniscono un valido mezzo per agevolare l'apprendimento degli argomenti.

Difficoltà: Alta • Autori: C.Beck, S.Dillon, J.Kallman, T.Kyte, H.Rogers
Editore: Apress http://www.apress.com • ISBN: 1-59059-286-7
Anno di pubblicazione: 2003 • Lingua: Inglese • Pagine: 1128
Prezzo: $ 49.99

GOOGLE, AMAZON, AND BEYOND: CREATING AND CONSUMING WEB SERVICES

Google e Amazon rappresentano, di fatto, due dei migliori siti presenti nel World Wide Web. Entrambi espongono alcune funzionalità che possono ritornare utili se implementate nelle proprie pagine Web. Gli autori illustrano come, utilizzando JAVA, XML, SOAP, REST e JavaScript, sia possibile invocare alcuni WebService di Google e/o Amazon, consentendo così ai propri siti di integrare funzioni evolute come: ricerca di libri, motore di ricerca, traduzione in diverse lingue, calcolatrice evoluta, ecc, ecc.

Difficoltà: Medio-Alta • Autori: T.Myers, A.Nakhimovsky • Editore: Apress
http://www.apress.com • ISBN: 1-59059-131-3 • Anno di pubblicazione: 2003
Lingua: Inglese • Pagine: 352 • Prezzo: $ 39.99

INBox
L'esperto risponde...

Control tab visual c++

Un saluto a tutti. Da poco (anzi pochissimo) ho inziato a programmare in visual c++, in passato ho programmato con Java, Delphi, qualcosa anche con C++ Builder, e vorrei chiedere il vostro aiuto per una faccenda. Volevo utilizzare un control tab per creare diverse interfacce sulla stessa form, ma dopo averlo inserito nella form non riesco a inserire in questo nessun controllo. Ho provato con delle edit ma restano nascoste sotto il controltab. Oltretutto non riesco neppure a cambiare la pagina corrente del control tab: come devo fare? Vi ringrazio

spartaco82

Risponde la redazione con johnkoenig

Il controllo a schede è un po' particolare: devi innanzitutto realizzare tutte le schede (come dialog box) stando attento alle dimensioni. Nella control tab, poi, devi gestire i pulsanti e le convalide. In ordine:

- Per ciascuna scheda nel foglio delle proprietà, creare un modello di finestra di dialogo e definire i contenuti e le caratteristiche della scheda. Impostare il titolo della finestra di dialogo in modo che questo venga visualizzato nel tab della scheda.
- Per ciascuna scheda nel foglio delle proprietà, derivare un classe simile a quella delle finestre di dialogo da *CPropertyPage*, che include membri di dati pubblici collegati ai controlli della scheda mediante DDX e DDV.
- Derivare una classe di una finestra di dialogo da *CPropertySheet*. Creare un'istanza della classe del foglio delle proprietà e delle classi della scehda del foglio delle proprietà derivate durante la fase 2. Utilizzare *CPropertySheet::AddPage* per aggiungere schede al foglio delle proprietà nell'ordine in cui si desidera che vengano visualizzate.
- Richiamare la funzione *DoModal()* dal foglio delle proprietà per la loro visualizzazione.

```
class CPrimaPagina : public CPropertyPage
{ public:
CPrimaPagina () : CPropertyPage(
                        IDD_PRIMAPAGINA) {};
// Dichiarare qui i membri di CPrimaPagina
protected:
virtual void DoDataExchange
                        (CDataExchange*); };
class CSecondaPagina : public CPropertyPage
{ public:
CSecondaPagina () : CPropertyPage(
                        IDD_SECONDAPAGINA) {};
// Dichiarare qui i membri di CSecondaPagina
protected:
virtual void DoDataExchange
                        (CDataExchange*); };
class CMioFoglioProprietà : public
                        CPropertySheet {
public:
CPrimPagina m_PrimaPagina;
CSecondaPagina m_SecondaPagina;
//Il costruttore aggiunge le pagine in
                        maniera automatica
CMioFoglioProprietà (LPCTSTR pszCaption,
              CWnd * pParentWnd = NULL) :
CPropertySheet (pszCaption, pParentWnd, 0)
{ AddPage(&m_PrimaPagina);
AddPage(&m_SecondaPagina);} };
```

Adesso, all'interno della *View* del progetto è sufficiente aggiungere le seguenti dichiarazioni:

```
CMioFoglioProprietà mProp(_T("Titolo del
                              foglio"));
mProp.DoModal();
```

Nuovo oggetto in Java: Numero Indefinito

Salve a tutti.
Vorrei creare un nuovo tipo di oggetto che rappresenti un numero (positivo o negativo) costituito da un minimo di una cifra ad un massimo arbitrario (credo anche superando i limiti imposti dai tipi primitivi) implementando anche le principali operazioni aritmetiche. L'unica soluzione sembrerebbe creare un array in cui memorizzare le cifre singolarmente, ma in quel caso le operazioni come la divisione sarebbero troppo complesse da creare. Quale potrebbe essere la soluzione?

Risponde kyakan

Per la serie "Pensa una cosa? In Java c'è già!" la classe che vuoi realizzare esiste e ssi chiama *BigInteger*, è un semplice oggetto *Number* che permette di gestire Interi di grandezza arbitraria. Questo oggetto fornisce non solo le convenzionali (e basilari) operazioni sui numeri (+, -, * e /) ma anche le operazioni orientate ai bit (&, |, ~, ^). In ogni caso generalmente questo tipo di oggetti viene realizzato mediante *LinkedList*. Naturalmente puoi implementare tutto con degli array, ma la soluzione (pur essendo non male) presenta alcuni problemi. Allora la più semplice che mi viene in mente può essere quella di creare una classe *Numero* che contiene un array con un elemento per ogni cifra.

```
public class Numero
{ protected boolean negativo;
  protected byte[] cifre;
  public Numero( Numero i )
  {       negativo = i.negativo;
          value = new byte[i.value.length];
     System.arraycopy(i.value, value);}
  public Numero( int i ) {
    if ( i<0 ) negativo = true; else negativo
                                   = false;
    value = new byte[Math.log(Math.abs(i))
                          /Math.log(10)];
    for ( numCifre = 0, i = abs(i);
                     i != 0; i = i / 10 )
    {  numCifre++;
      value[numCifre] = (byte)(i % 10); }
  } ...
}
```

Fornendo un valore come *String* viene convertito in numero, naturalmente bisogna fornire anche il costruttore a partire da *int* o *Integer*.
La somma puoi realizzarla tenendo conto della dimensione del numero, ed è per questo che una Lista lincata era meglio infatti ridimensionare una lincked list è costante mentre per un array ha costo $O(n)$. La moltiplicazione merita invece un algoritmo più efficiente, ne mostro uno non proprio banale. È noto che una moltiplicazione di due numeri a n cifre essere vista come una somma ripetuta il che porta ad una complessità quadratica rispetto alle cifre $O'(n^2)$. Se prendi due numeri *A* e *B* e vuoi moltiplicarli: *A * B* puoi riscrivere questa moltiplicazione nel seguente modo $A*B=X1+(X1+X2+X3)*10^{(n/2)}$ $+X2*10^n$ dove *X1=A1*B1,X2=A2*B2, m3=(A1 – A2)*(B1–B0)* e quindi il calcolo del prodotto è ridotto a solo tre prodotti a cifre dimezzate e alcune somme che hanno costo lineare (come anche la scomposizione). Questo algoritmo ha quindi una complessità ridotta pari a $O'(n^{ln(3)})$ che è migliore per grossissimi valori di n (a piu cifre). Ciao!

SXPER CONTATTARCI
e-mail: iopinbox@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano

Combinare oggetti

di Fabio Grimaldi

Il calcolo combinatorio in generale e le combinazioni in particolare, rivestono un aspetto fondamentale nella programmazione; a questo si aggiunge un'interessante quanto intricato percorso per la soluzione di alcuni problemi legati a tale ambito.

Gironzolando per il forum di ioprogrammo *(www.ioprogrammo.it)* mi sono imbattuto in uno stuzzicante intervento nel quale, l'ormai noto problema del giro del cavallo, (che prosegue la sua attività di ricerca del percorso, aiutato da tanti programmatori di buona volontà che cooperano e sviluppano anche grazie al forum e allo spazio apposito messo a disposizione dalla rivista sulla rete), veniva affrontato con algoritmi genetici. In particolare *caino*, questo il nick dell'ideatore, proponeva un intrigante parallelo con algoritmi per la comparazione di sequenze di DNA. Senza dilungarmi, si suggeriva di affrontare il problema segmentando le soluzioni. In una prima fase si generavano tutti i possibili percorsi di lunghezza quattro del cavallo a partire da una qualsiasi casella, azione assimilabile alla ricerca di filamenti del DNA di lunghezza quattro. Prima di proseguire vorrei subito far notare come si tratti di un puro problema di calcolo combinatorio. L'algoritmo proposto, prosegue iterando la ricerca a percorsi di lunghezza 8, poi 16 ed ancora 32 fino ad arrivare a 64 che centra l'obiettivo preposto. Ovviamente, ogni soluzione parziale tiene conto dei risultati ottenuti ai passi precedenti. Secondo le note dell'autore il problema non riscontrerebbe complessità esponenziale; questa ed altre sono le questioni inerenti il calcolo combinatorio a cui ci interessa dare risposte. Nel presente articolo verranno proposti due fondamentali interrogativi, entrambi inerenti le combinazioni.

PREMESSA

È risaputo che il calcolo combinatorio è quel particolare ambito della matematica che studia il raggruppamento e la classificazione di oggetti. I raggruppamenti possono distinguersi, oltre che per gli oggetti coinvolti, sostanzialmente in funzione di alcuni altri parametri. Il più importante tra essi è l'ordine. Nel particolare raggruppamento, conosciuto come disposizioni, un carattere distintivo è l'ordine, mentre le combinazioni non fanno differenza tra gruppi di stessi oggetti posizionati in ordine dissimile. Infine, le permutazioni sono particolari disposizioni in cui non esiste un'ampiezza di classe di raggruppamento. Un secondo parametro è la possibilità o meno, di ripetere gli oggetti da sistemare. Chiariamo le idee con un piccolo esempio. Supponiamo di avere cinque palline di diverso colore: blu, nero, rosso, giallo e verde; che indicheremo con: *b, n, r, g* e *v*. calcolare le disposizioni dei cinque elementi *a* classi di raggruppamento di ampiezza due significa applicare la formula: *D5,3=5*4=20*. Ecco le venti disposizioni:

- b, n - - b, r - - b, g - - b, v -
- n, b - - n, r - - n, g - - n, v –
- r, b - - r, n - - r, g - - r, v -
- g, b - - g, n - - g, r - - b, v -
- v, b - - v, n - - v, r - - v, g -

Come si può notare alcuni raggruppamenti differiscono solo per l'ordine in cui si presentano le palline, ad esempio *-b,n-* e *-n,b-*. La generalizzazione della formula appena descritta per n elementi della classe *k* (gruppi di *k* oggetti)è: *Dn,k_=n*(n-1)* ...*(n-(k-1))*. Le combinazioni sono in numero minore delle disposizioni. Con riferimento all'esempio esaminato non sono gruppi diversi *-b,n-* e *–n,b-*). La formula che ne calcola il numero esatto è: *Cn,k=n!/(n-k)!*k!* Il punto esclamativo indica il fattoriale. Nel caso di cinque elementi della classe due risulta *C5,2=10.* Qualora nei raggruppamenti esposti: disposizioni, permutazioni e combinazioni, gli elementi si ripetono (ad esempio, le palline si possono estrarre più volte di eguale colore) allora, siamo di fronte a raggruppamenti con ripetizione il cui calcolo del numero varia rispetto a quello presentato.

Problema 1 - Si vuole definire un metodo che, per qualsiasi *n* e *k* sia in grado, oltre che di calcolare il numero di combinazioni (basterebbe applicare la formula), di individuare le singole combinazioni. Sembra banale ma come intuirete, così non è. Subito si riscontrano applicazioni pratiche, ecco che il problema prima citato, del giro di cavallo con algoritmo DNA, richiederebbe qualcosa di simile.

Problema 2 - Un'ulteriore richiesta è quella di scegliere tra tutte le combinazioni in base ad altre specifiche. Ad ogni oggetto è associato un valore e dei vincoli. Si vuole trovare una combinazione (in alcune formulazioni del problema, anche con ripetizione) che massimizzi il valore ottenuto nel rispetto dei vincoli imposti dal problema. Supponiamo un attimo di essere dei furfanti, ladri in particolare, con tanto di borsa di dimensioni ben precise. Ogni oggetto che possiamo trafugare ha un valore e delle dimensioni, si vuole sapere quale combinazione di oggetti massimizza il furto. Si tratta del problema Knapsack.

CONCLUSIONI

Come si vedrà la prossima volta anche il problema della dieta può essere formulato in questo modo. Vi saluto e vi lascio alla soluzione dei singoli problemi proposti.

SOLUZIONI AI PROBLEMI DI IOPROGRAMMO 76.

SOLUZIONI

N. 1 - Utilizzando delle tessere di domino della grandezza di due caselle di una scacchiera, veniva chiesto di trovare una disposizione che coprisse tutta la scacchiera tranne le due case posizionate nei due angoli opposti. Le risposta corretta era da individuare tra la possibilità di usare 31 tessere, che coprono esattamente 62 caselle che è il numero richiesto e la possibilità che non ci sia una disposizione che permetta l'intera copertura. Facendo qualche prova ci si rende subito conto che siamo di fronte alla seconda eventualità, poiché le caselle occupate negli angoli rendono dispari il numero di caselle sui lati, il che rende impossibile la contemporanea sistemazione di caselle lungo lati adiacenti.

risposta esatta: ***d)*** non è possibile riuscire a coprire la scacchiera nel modo richiesto.

N. 2 - Usando un misuratore di distanze di stelle dalla terra, che è in grado di individuare la più vicina tra tre, si vuole sapere qual è il numero minimo di misurazioni per trovare la più vicina tra 99. Come avevo scritto è abbastanza facile. Con le prime 33 misurazioni si restringe la ricerca a 33 stelle, dividendo sempre a gruppi di tre si effettuano altre 11 misurazioni che lasciano 11 stelle da esaminare. Delle rimanenti, nove vengono osservate con tre misure. Rimangono 5 astri. Una ulteriore misura restringe il campo a soli 3; un'ultima misura ci porta al risultato. Facendo i conti (33+ 11+ 3+ 1+ 1=49) si ottiene 49. La soluzione si attuava anche con l'uso di un albero.

risposta esatta: ***b)*** 49)

N. 3 - Sull'esercizio di logica basta analizzare le singole frasi e stabilire qual è verificata rispetto all'asserzione iniziale. Ad esempio, la seconda affermazione non è corretta perché, se è vero che tutti i belli sono ricchi, non sempre è vero il contrario, ossia, ci può essere qualche ricco che non sia bello.

risposta esatta: ***a)*** alcuni tristi sono belli.

N. 4 - Il problema dalla affascinante formulazione, tratta i sistemi di numerazione. È noto che il sistema decimale, su cui si fonda la nostra algebra, sia stato adottato in passato (studi dimostrerebbero che fu adottato per primo dagli arabi) in relazione alle nostre dieci dita. Per cui, se l'operazione ritrovata su marte, 4+5=11, ci induce a pensare che sia usato un altro sistema di numerazione, che è l'ottale. I marziani hanno quindi quattro dita per mano.

risposta esatta: ***a)*** 4.

N. 5

Adottando una tabella di verifica si deduce che entrambe le funzione calcolano la somma. La seconda è il risultato della famosa serie di Gauss. Un aneddoto circa il bambino prodigio Gauss dice che, quando frequentava le scuole elementari, il maestro dovendo fare una commissione, avesse assegnato agli alunni un problema che li tenesse impegnati per lungo tempo. Si trattava di fare la somma dei primi 100 numeri. Il maestro però non fece neanche in tempo ad uscire dall'aula che il giovane Gauss esclamo 5050. Il genio fece questa considerazione, sommo i due estremi della successione, 1 con 100 è ottengo 101, poi sommo 99 con 2 e il risultato è sempre 101. Il 101 (n+1) si presenta così esattamente 50 volte (n/2), ecco che si ottiene la formula della seconda procedura.

risposta esatta: ***b)*** sia la funzione ***A*** che la funzione ***B*** calcolano la sommatoria di tutti i numeri compresi fra 1 ed ***n***, assumendo ***n*** maggiore o uguale a 1.

L'ANGOLO DELLA COMPETIZIONE

NUOVI PROBLEMI

1) - Un ladro riesce ad aprire la cassaforte di una banca dove si trovano 100 sacchetti pieni d'oro, ma ha la possibilità di rubarne uno solo. 99 sacchetti hanno il medesimo contenuto mentre uno è più pesante in quanto contiene più oro. Chiaramente il ladro vuole portare con sé il sacchetto più pesante. Ha a disposizione una bilancia a due piatti: su ogni piatto può mettere la quantità di sacchetti che vuole, e la bilancia gli indica il piatto su cui è collocato il peso superiore.
Prima di iniziare a pesare i sacchetti il ladro vuole calcolare quante pesate dovrà al massimo fare per avere la certezza di scegliere il sacchetto giusto. Quante sono queste pesate?

Risposte:

a) 6
b) 7
c) 99
d) nessuna delle precedenti

2) - Si consideri il seguente frammento di programma:

```
Linguaggio PASCAL:
type
        vettore = array [1..10] of integer;
        vettore2 = array [1..20] of integer;
procedure quack(var a: vettore; var b:
          vettore; var c: vettore2; l: integer);
var
    i, j, k: integer;
begin
    i := 1;
    j := 1;
    k := 1;
    while k <= 2*l do
    begin
        if ((a[i] < b[j]) and (i <= l)) or (j =
                                        l+1) then
        begin
            c[k] := a[i];
            i := i+1;
        end
        else
        begin
            c[k] := b[j];
            j := j+1;
        end;
        k := k+1;
    end
end;
```

Qual è l'output?